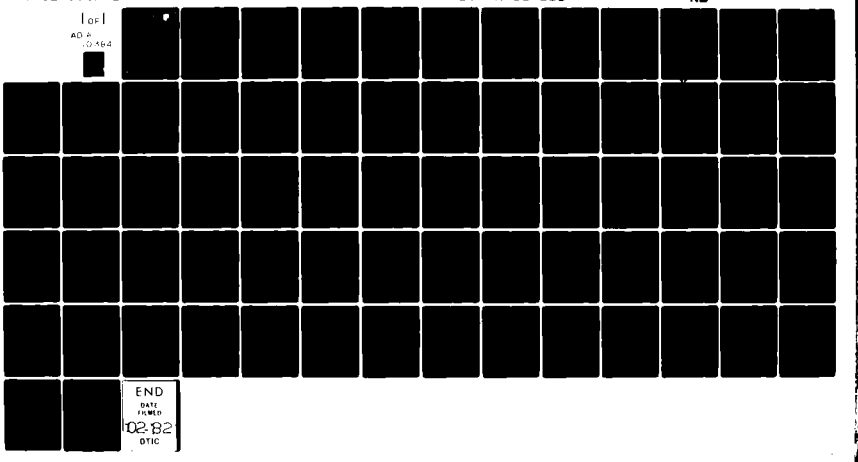
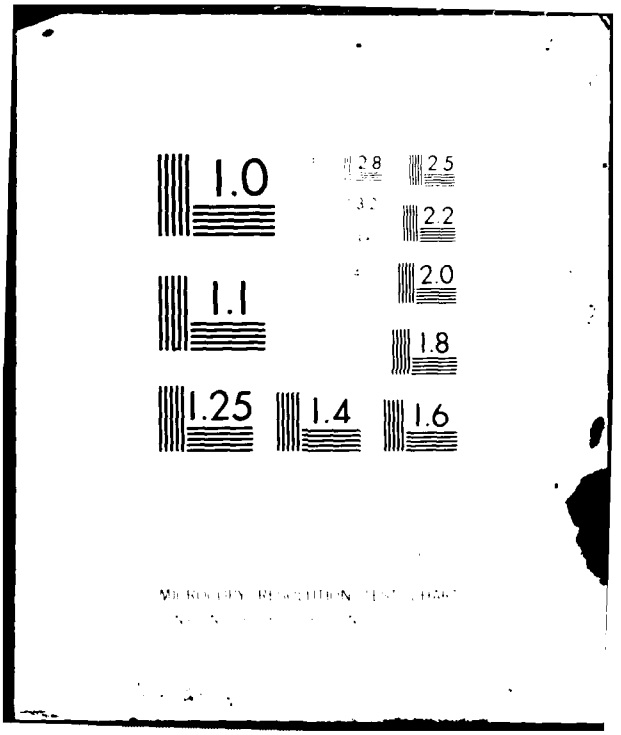
AD-A110 384 METAGRAM NEW YORK F/G 9/2
METAGRAM SOFTWARE - A NEW PERSPECTIVE ON THE ART OF COMPUTATION--ETC(U)
OCT 81 D G HAYS, W L BENZON F30602-80-C-0180
UNCLASSIFIED RADC-TR-81-118 NL



END
DATE FILMED
02-82
DTIC


1.0
2.8
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

RADC-TR-81-118
Final Technical Report
October 1981

LEVEL II

12

METAGRAM SOFTWARE - A NEW PERSPECTIVE ON THE ART OF COMPUTATION

METAGRAM

David G. Hays
William L. Benzon

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-118 has been reviewed and is approved for publication.

APPROVED: Zbigniew L. Pankowicz

ZBIGNIEW L. PANKOWICZ
Project Engineer

APPROVED: John N. Entzminger

JOHN N. ENTZMINGER
Technical Director
Intelligence and Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-118	2. GOVT ACCESSION NO.: AD-A110 384
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): METAGRAM SOFTWARE - A NEW PERSPECTIVE ON THE ART OF COMPUTATION	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report 1 May 80 - 31 Oct 80
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): David G. Hays, William L. Benzon	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS: METAGRAM, 150 Fifth Avenue, New York NY 10011	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 31025L, 96820152
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDT), Griffiss AFB NY 13441	12. REPORT DATE: October 1981
	13. NUMBER OF PAGES: 72
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Zbigniew L. Pankowicz (IRDT)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Programming
Metagramming Philosophy
Abstraction & Metasystems
Applicative Computation Theory
Software Management
Information and Analysis
Intelligence Information Systems
Metagramming Implementation
Cognitive Science
Gibsonian Psychology

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The report documents the results of a six-month R&D effort consisting of a critical examination and feasibility test of the metagramming technique to assess its innovative utility in providing an improved access to databases in the COINS network. The introduction briefly describes current problems in software development/management and outlines metagramming principles. The first chapter illustrates state-of-the-art limitations of conventional programming. The second chapter elucidates

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the conceptual foundation of metagramming (multi-level abstraction, cognitive processes) and describes a three-level computational system based on metagramming. The third chapter discusses a continuous evolutionary growth of cognition to progressively higher strata described as a sequence of cognitive jumps, each of them characterized by a greater control over complexity than its predecessor. The historical evolution of computational technology is described in the fourth chapter, prior to highlighting the role of higher-level abstractions and the "universal executive" inherent in the metagramming strategy of computation. The fifth chapter envisions the development of metagramming technology as a series of successively easier-to-use machines. The problem of control in metagramming processes is addressed in the sixth chapter. The seventh chapter discusses computational requirements associated with progressively more complex world models inherent in the evolution of metagramming from the initial system (level 0) to a multi-system (level 6). The last chapter deals with the applicability of metagramming to intelligence needs as a means of substantially enhancing the analytic competence of the intelligence community. A discussion of metagramming in the context of intelligence requirements is provided in the Appeddix.

SUMMARY

Metagramming offers a new perspective on information systems. Its aim is to achieve conceptual control at a level that would reduce the software problem that slows the growth of computation. Such a level of control would also serve well in the analysis of military and political intelligence, and in other areas where highly abstract methods of thought serve practical ends.

Contemporary methods of programming include at least two levels of abstraction. One is the standard level of instructions, subroutines, and so on, conceptually very closely related to the fetch-execute cycle. The higher level is exemplified by a few systems such as LISP and FORTH, which provide the user with a concept of functions. These systems include executives which define the more abstract level in relation to the lower.

Metagramming concerns itself with the executive. A universal format for executives constitutes a model for the relation between a more concrete and a more abstract system. This is the metarelation.

The evolution of human culture can be interpreted as a sequence of stages, each with new metarelations establishing control over the fundamental processes of the prior stage. All of the previous stages are represented in the contemporary population; computer software, military command systems, and other practical affairs must meet the condition that persons with varying levels of cognitive power have to operate their several components. Hence the problem of software design is the use of metasystems to construct worlds (databases, computing languages) adapted to the various levels of users. The present lack of intermediate grades, between the universal and the specific (between FORTRAN and word processors, say) forces many persons to operate in worlds that suit them badly, reducing their productivity and inducing error.

Metastructures are needed in intelligence because several levels of abstraction are involved in a political or military system, because analysis entails a complex interplay between presuppositions and conclusions, and because international relations is an intricate mixture of mutual trust and mutual deceit.

The initial implementation of metagramming systems requires neither a large expense nor a long period of development. Even the initial systems should have practical value. If this proves to be the case, then benefits should increase with time.

PREFACE

For some readers, the Appendix may provide the most convenient entry to the report. It contains, in particular, an explanation of the necessity for dealing with a wide range of matters in the process of coming to grips with the software problem.

Several persons have contributed to the development of the metagramming concept. William L. Benzon, in addition to writing the Appendix, worked on the evolutionary sequences; the idea of ranks comes from his earlier writings, but is somewhat extended here. The sequence from tools to machines is his. The crucial idea of a functional form with four slots is due to Richard Fritzson. Michael S. Howard supplied specifications of requirements for implementation. John W. Carr III reviewed work in progress in several helpful sessions. David S. Wise supplied a clear understanding of current work in applicative programming.

CONTENTS

INTRODUCTION

Military intelligence and computer software deserve the attention of every citizen. From May 1 through October 31, 1980, Metagram examined these topics, separately and jointly, from a new perspective. Each topic is subtle. Each has great practical importance to the citizen and the state. Each poses problems that have proved difficult to formulate, let alone solve. Metagram concludes that familiar perspectives conceal from the observer formulations that can be both elegant and effective.

To adopt a new perspective is no easy matter.

To bring the reader to a perspective from which new modes of computation and of intelligence analysis can be recognized is the chief aim of this report. Enough must be said about present conditions to supply a reason for drastic change. A brief account of some revolutionary changes in systems of thought may enhance the plausibility of the proposition that perspectives can change. An outline of the modes of interaction with computers as it may be if metagramming is generally adopted provides at least a hint of the value that the new perspective could have. And a technical sketch of a fundamentally new method of information processing gives some substance to the argument.

The generalist, rather than the specialist, is the reader for whom the report is written. The specialist who does not see his own specialization with the ironic eye of the generalist is fettered by the very schemes that enable him to work effectively. The fetters of the software specialist create the software problem that we face today. Metagram offers its methodology as a natural successor to programming and to analysis as the art is known in the discipline of intelligence. The specialist who is ready to discard the fetters and look at the computer in a new way can find help here.

THE SOFTWARE PROBLEM

The condition of the software craft today is not good enough. High cost, tardiness, error, and inflexibility are problems known to everyone who has anything to do with computers. These problems arise in every sector: In military operations, in all of government, in industry, commerce, science, and education. Software problems retard development of systems, stunt the growth of the market, and induce operational failure and dissatisfaction wherever the computer is used. Even the distribution of computers for personal use is affected. These facts are too well known to require citation of specific reports, which have been numerous in administrative, technical, and popular media. The software problem is, and is widely known to be, grave.

No solution commensurate with the problem has been suggested. Some have proposed or attempted techniques of planning that exacerbate the problem of tardiness; techniques of documentation that introduce new loci for error; techniques of translation from new languages into the very languages which,

when introduced, were supposed to be readable and writeable but now have no virtues to justify them as targets for program translation. To hyperbolize a cliche, the corpse is mummified with Band-Aids, but still lies stinking inside all of the contemporary proposals.

Moreover, the software problem bars a crucial path. The computer is the most advanced tool ever built, and at the same time offers greater promise of further enhancement than any other. It is a toy. It is a tool of production, management, and planning. It is an aid to understanding, and with better understanding we expect to solve enough of our problems to survive and perhaps even to make the world more secure and satisfying. It is an aid to understanding, but the irony--no doubt the inevitable irony--is that we do not understand the computer. And that is the software problem.

For twenty years, understanding of the substance of the computer--how to make larger, faster, cheaper machines--has grown, while understanding of the computer's function has grown only very slowly. Great benefits have been lost, despite great efforts to sieze them; talent and support have not solved the problem. Surely nothing easier than enlargement of our perspective on the whole affair will ever solve it.

THE INTELLIGENCE PROBLEM

Metagram has had access only to the most public kind of information about the intelligence community and its activities--what anyone can learn by browsing in a major library. To the outsider, it seems likely that the work of intelligence must be susceptible of improvement. The situation is, and not by coincidence, like that in software. Problems of cost, timeliness, accuracy, and flexibility must exist. Some small evidence to this effect is easy to find, but the value of the evidence is uncertain. What is more certain is a comparison.

The computer is a tool for understanding; and understanding is the function of intelligence. The making of software is a task of conceptualization, and because it is a conceptualization of understanding, it is a task of a very high order. The analysis of intelligence is also a task of conceptualization, and because it is a conceptualization of the mode of thought and operation of an opponent it is a task of comparably high order. That is to say, the programmer must rationalize the way WE think, so that the computer can serve us; the analyst must rationalize the way THEY think, so that their acts can evoke effective response from our side.

The level of thought required for intelligence analysis, like that required for computer programming, is at the very limit of our culture's ability to formulate and manipulate concepts. To bring into the arena of practical everyday affairs methods of thought that lie at the culture's furthest limits is a great risk. But we could not eliminate the computer from practical affairs on account of the risk; the cost that we would surely have to pay would be too great. And for similar reason we could not reject the most advanced methods of thought from the practical affairs of military intelligence because we would certainly pay a high cost in eliminating the risk.

Once the programmer delivers a rationalization--a program--the computer does much useful work; the difficulty in making the rationalization is what we know as the software problem. Once the analyst delivers a rationalization--a scenario or other estimate of THEIR policy--methods now widely familiar give many useful results (war gaming and the exploration of complex mathematical models are examples); the difficulty that surely inheres in the formulation of scenarios can appropriately be called the intelligence problem. Given the perspective of the best thought in our culture, computers are useful and programming is hard. Given the analogy between computation and intelligence, that same perspective would make gaming useful and analysis hard.

What is at stake in analysis of military and political situations is important. In gross terms, a nation that arms in a peaceful world is insane, but a nation that abandons its arms in a hostile world is equally so. Between these insane extremes spreads a vast range of adjustments of amount, of kind, and of placement. Slightly too much, slightly too little, or not quite in the necessary location need not be an insane error, but can have serious consequences nevertheless. In a large conflict, better judgment can contribute to victory in a way that better armaments cannot.

The judgment of the adult is better than the judgment of the child, because the adult has a perspective that the child cannot adopt. The judgment of a person with adequate training in modern systems of thought--scientific, for example--is better than the judgment of a person who cannot adopt such a perspective; we rely on the judgment of physicians for that reason. The thesis of this report is that a new perspective is becoming accessible.

METAGRAMMING

The assertion that metagramming offers a new perspective on computation is a way of claiming for it a new degree of conceptual power. This kind of claim cannot be proved, but it can be clarified and illustrated.

In plain language, a system with greater conceptual power can think or talk about what a system with less power can only do. The more powerful system can thus describe, or in some limited sense understand, the conceptually weaker system. It has concepts of, names for, patterns of concepts that occur in the simpler system. For almost everyone, the more powerful system is also more convenient to use, since it can relieve the user of responsibility for detail. The bare computer is an "idiot machine", but programming and metagramming can use concepts of higher degree.

Even the bare computer is more powerful than the data it processes. The crucial concept is the conditional jump. This remarkable operation can recognize a pattern in the data and change the flow of execution through the program accordingly. The conditional jump adapts the program to the changing stream of data.

A very few advanced systems of programming are more powerful than the computer. Typical languages (FORTRAN, PASCAL, etc.) are convenient, but their organization is of the same general kind as that of the elementary computer.

LISP, growing machines (FORTH), and functional programming are different. Functions are concepts for patterns of machine operation; instead of conditional jumps, the functional or applicative programmer uses conditions on the application of functions. To link the more powerful scheme back into the computer, which must make its conditional jumps even when running Lisp or Forth, a translator is required. It may be quite small, but it must use a sophisticated trick: It calls itself recursively to manage a nest of functions. The power of functional programming shows up in the compactness and clarity of the code and the productivity of the writer.

The next grade of power comes with adoption of the executive as the basic unit of computational structure. With multiple executives, piled one on top of the other, systems more and more remote from the basic computer can be made to run. If a function is a concept for a pattern of machine operation, then at the next level one has concepts for patterns of functions; call them models. The next level again has concepts for patterns of models. And so on. Clearly, if the system can maintain its own internal coherence, a user who merely orders the application of a model to a problem will be more productive than one who has to specify all of the functions involved.

Metagramming is the art of making and assembling executives. But the making of an executive is a major task; very few executives have been invented in the history of computation. Toward simplifying the task, Metagram offers a schematic form and a basic mode of operation. To make a new executive, one fills in the form.

What metagramming offers to most users of the computer is not the full power of a highly abstract system, but the convenience that can be provided by designers who have that power available to them. FORTH is powerful; but the user of a text editor written in FORTH need not even know that the editor is good because the power of the system encouraged someone to design a it so. A metagramming facility should encourage designers to create new systems for different classes of users, variously offering ease of use for special purposes or great capacity for broad purposes.

SOFTWARE PROBLEMS

Even the purchaser of a home computer can presently have access to many languages and data bases; the computer can be plugged in to a telephone line, and suppliers can be reached by dialing. Commercial, educational, and governmental users have access to a much larger variety of sources. Yet the diversity is not so pleasing, since each source imposes burdens of memory and skill on the user; no one can afford to learn the vocabulary and grammar of every programming or database access language. Some illustrations follow. They are mostly from a system that may have been improved by this time; Metagram does not have current information. No particular criticism of the authors is intended. Many systems of comparable difficulty are in use; it is the condition of the whole field that needs attention.

REDUNDANCY AND INTRICACY

To sign on, according to a users' guide, requires several steps. The words in parentheses are names of special keys; 'H' is the human user; 'M' is the machine.

```
H (RESET)
M 0715 A QRS READY Q00015
H (ALT MODE) (CTRL, SVC)
H T115
M T 115
H (ALT MODE) (CTRL, SMK)
M *
H 3,ABC,115
H INTG
H SMITH,A22,3456
```

Note that prompts are rare; this signon is followed by a request, and nothing comes back from the machine until the request is finished. Then, and only then, the user may learn that the machine has encountered an error.

The special keys are, perhaps, only a hardware problem. 'SVC' means service, and 'SMK' means send a message from the keyboard. Hardware of this level imposes a burden on the user; a better level is certainly available now.

The machine's first line is not especially objectionable. '0715 A' is the time; 'QRS' is the system, which should identify itself. 'Q00015' is the present user count, which permits an estimate of response time; but the number is in octal, not decimal.

The next interchange is regrettable. The 'T' is a request for connection with a subsystem, and '115' identifies the terminal in use; the manual explicitly forbids any blank between; then the system echoes the line with the forbidden blank. Does the system not know what terminal is connected?

The prompt '*' acknowledges that a message is to be received from the user. The next line consists of priority '3', agency addressed 'QRS', and again the terminal identification '115'.

Without further prompt, the user enters 'INTG', meaning that the message is a file interrogation. On the next line, name 'SMITH', organization 'A22', and telephone number '3456'.

All this is not to be explained to a novice in a page or two, and in a situation where each potential user has access to many systems most users must be novices most of the time. A microcomputer terminal with a little memory and a cathode-ray tube display could simplify matters substantially. In a suitable language, a short and simple program could retain the protocols for signing on to various subsystems; store the terminal's own identification; keep a list of local users, with name, organization, telephone number, etc.; and store displays offering courses of action to the user ('menus') to which the user could reply with very few keystrokes. The burden on the novice or infrequent user could be almost eliminated.

CONFUSION OF LEVELS

The user should be able to get what he wants without knowing a thing about storage media and storage formats. The differences among tape, disc, and high speed memory; the widths of fields; ASCII or EBCDIC; and all such qualities of information systems are fascinating but irrelevant to the concerns of the personnel department, the intelligence analyst, and others with practical concerns. Nevertheless such considerations frequently intrude in the process of program design. If the user is not to know, the system must handle the details.

An example arises in a language that has been used for intelligence materials; Metagram does not assert that this language is in current use. The point is again not to impute incompetence or inadequacy, but to make clear the standards that prevail in computation.

> QRS files ... come in two varieties: single-format files and multi-format files. The latter are of recent origin to meet the demand for files with larger and more flexible records. Most files are, however, single-format files. As the name implies, all records from a given [single-format] file have the same fields, arranged in the same format, containing similar data. The only difference among records is the data recorded in it. In QRS the maximum character length of each record is 495, and the maximum number of fields allowed is 60.
>
> Multi-format files allow for larger records to be built, and give increased flexibility within a file. [Users Guide, page II-1]

A multi-format personnel file might contain, for each employee, "relatively permanent" characteristics in one record, with data on spouse in a second record, on job assignment in a third record, on education in a fourth, and so

on; but the illustration in the Users Guide stops here. A "group" of records describes an employee; and each record in the group has a different format.

> The user of QRS must know about multi-format files.
>
> From the standpoint of the user the difference in making a query in the two kinds of files is
>
> for a single-format file: specify file, field, field content;
> for a multi-format file: specify file, format name, field, field content. (Page II-3)

So, if the personnel file had a single record for each employee, with a field for language skill, the user could specify

> Personnel (file) Language (field) French (content)

and find all Francophone employees. But, assuming a multi-format personnel file, the user must instead specify

> Personnel (file) Education (format) Language (field) French (content)

To realize the severity of the problems that arise, the reader should think of a large corporation with a central office and many divisions or subsidiaries. Each subsidiary has its own personnel file and can decide for itself whether to use one or several records for each of its employees. The user is in the central office: A junior member of the president's staff, assigned to find a French-speaking secretary who can accompany a sales team to Paris. How much time will the staffer spend learning about the file structures? And how much learning about the personnel? The staffer goes from task to task, consulting different files from day to day. The time spent studying file structures can be serious.

> The INFO verb provides QRS users with information about any file currently available via the QRS language. This information is descriptive in nature; that is, it would normally consist of record length, the field names, sort fields, etc. ... A request of this type will yield output for each format in the file; thus, if the file is multi-formatted...this could be a lengthy print-out. (Page X-3)

But the user can ask what formats a file contains, and then ask for information about selected formats.

The QRS system will give the naive user an unexpected result from the query stated above. The Users Guide explains "Group Search" (pages IV-13 through IV-15). The human employee is represented in a multi-format file by a Group of records. Among the relatively permanent characteristics of an employee are name and address (page IV-13); but language skill is recorded in the education format. Social Security Number is the only datum common to all the records in the employee's group. So the naive user receives a list of Social Security Numbers of employees who speak French--because the naive

user writes the same verb for extraction of appropriate employees from single and multi-format files.

The trained user of QRS uses the verb EXTI (extract inclusive) for single-format files and EXTIG (extract inclusive group) for multi-format files.

One qualification on the use of EXTIG is that it is unnecessary and wasteful if the information needed is entirely within one format. If name and address are in the first format of the personnel file, then EXTI is suitable for an extraction by address to prepare, say, a mailing of invitations to the annual picnic. (On the other hand, if information about spouse is needed, then EXTIG must be used after all, since Spouse is the theme of the third format.)

QRS has boolean connectors AND and OR. If the sales team is going to both Paris and Berlin, the query from the president's office might be

```
EXTIG Personnel (file) Education (format) Language (field)
French AND German (content)
```

The response is a list of employees with skill in both foreign languages. Now, if company policy is to send only male employees abroad with sales teams, the search must be restricted by sex. But sex is certainly a permanent trait and would appear in the first of the multiple formats (the file is hypothetical, and the Users Guide does not mention this trait). The request must be

```
EXTIG   Personnel (file)
        Basic (format) Sex (field) Male (content)
        AND/REC
        Education (format) Language (field) French (content)
```

(The conjunction of German would be admissible.) The difference between AND and AND/REC belongs to the level of file design; conceptually, both are simple conjunctions of terms.

In fact, the QRS request would be entered more compactly. The lines beneath the example indicate its syntactic organization; they are not part of the request.

```
EXTIG/PER;FORM(AA) AND SEX(MALE) AND/REC FORM(DD) AND LANG(FRNCH)
          --------++++---------         --------++++-----------
          ------------------------++++++++-----------------------
      ---+-------------------------------------------------------
------+-------------------------------------------------------------
```

Each AND connects a pair of terms, one specifying format and one specifying field(content); the underscoring shows the scope of each. The AND/REC connects the two composite terms. The semicolon connects filename with specification. The shilling or slash connects verb with object. The spacing, which seems arbitrary and counterintuitive, is necessary to the system.

The difference between field and format has undergone a metamorphosis. In the file, FORM is itself a field; so the Education format is identified by content DD in field FORM. The user must write a conjunction between FORM(DD) and LANG(FRNCH) as if these terms were on the same level; and indeed they are on the same level in the file, although conceptually they are incomparable.

The information that the user must take into account in choosing EXTI or EXTIG, in choosing AND or AND/REC, in deciding to include FORM, and in specifying fields is all available to the system. A microcomputer as the user's terminal could solicit file structure from the INFO file. It would be able to take, say, FRNCH and determine that the field must be LANG. The microcomputer could determine, from the content of INFO, that SEX and LANG are fields in different formats, and select AND/REC as the conjunction. It could even remember that the employee's name is often wanted and ask the user whether to extract by group, obtaining the name field, or to conserve system time and lose the name.

WIDER PROBLEMS

As the foregoing examples suggest, the syntax and vocabulary of a language for computation is all too often adapted to the structure of the machine. The implementation, rather than the application, determines the structure of the language. Punctuation and spacing are designed to simplify the program that interprets input, however unnatural the design may be for the user. The logical structure that the user must concoct intermixes relationships from the user's domain with relationships from the implementation, such as the format of records.

Programming deals largely with repetitive tasks. The serious user has to construct a program that controls repetition, although database languages generally manage this aspect of the work automatically. Even such obvious purposes as "Shift each character on the line to upper case" or "Add up the earnings of all the employees in the file" turn out to require thoughtful analysis; programmers must acquire considerable skill if they are to manage the repetitiveness of computation accurately, and even skilled programmers make errors.

The systems that experts use give them greater freedom of action than the systems used here for exemplification. But the experts' systems also tend to demand even greater attention to detail. The broadest statement of the problem that hinders the work of almost every computer user is that the systems and languages of present-day computation are too closely bound to the level of the machine itself--to its primitive representation of data and to its modes of operation.

ABSTRACTION AND METASYSTEMS

Proverbs, poems, and puns often indicate the form of ideas that we cannot convey in words. Explaining a proverb is a difficult task; children fail entirely, and adults often feel that the explanation loses the point, just as the explanation of a joke loses the humor. The point of a good proverb is abstract, and abstraction itself is one of the ideas that we can indicate but, as yet, not convey adequately in direct exposition. Metagramming is an idea about abstractions and their manipulation.

REDUCTION OF INNUMERABLE DETAIL

Abstraction, like generalization, simplifies thought. Whereas generalization eliminates detail, abstraction organizes detail into a pattern. The statistician who calculates the average of a collection of data is making a generalization; the one who announces that the data come from a normal distribution is making an abstraction, that is, seeing a pattern in the material. Eliminating details of distortion from many views of a horse, one can capture the generalization that it is the same horse; organizing those details, one can capture the abstraction of a gallop--the horse, seen in many consecutive views, is galloping. The human nervous system can form perceptual abstractions: Any normal person can see both gallop and horse. Other kinds of abstraction require more than growth of the nervous system in an ordinary perceptual environment; they are provided by culture.

At a very different level, an isolated case presented to a computer for processing is like one view of the horse, and by generalization a file of such cases can be reduced to a common pattern: They are all payroll entries, or database records in a given format. Many simple computations can be understood very well in this way. When a program consists of ten or a hundred thousand lines of code, generalization ceases to provide understanding. The horse is galloping, and only an abstractive method can make the process comprehensible.

Sheer magnitude seems to be involved. Eliminate details from what is seen to make a generalization; give a name to the generalization and give names to the details. If the details are few, a syntactic construction of the names of the details (say as adjectives) and the name of the pattern (as noun) makes an understandable utterance (a noun phrase). But if the details are numerous, naming them makes the utterance incomprehensible. Abstraction of a pattern from the numerous details reduces the expression to simplicity again. Thus a person who could neither read nor write a suitable computer program can speak effectively of the payroll program and its place in the management of the company's affairs. A person without knowledge of integral calculus can use the concept of area successfully.

Magnitude of detail suggest a formal explanation of abstraction as any function with a sufficiently large set of arguments, or, idealizing, with an

infinite argument set. The integral calculus is perhaps the earliest example of mathematical success with abstraction; integrating is summing indefinitely many terms, each indefinitely small. By integration, one passes from the line to the more abstract plane, from speed to the more abstract distance; before the calculus was invented, attempts to compare speed and distance across levels of abstraction yielded only paradox. At present, attempts to understand perception without recognizing that it covers several levels of abstraction lead to paradox. The gallop one sees is a perceptual function with a very large number of arguments, all of the distortions of the horse and the changes of distortion from moment to moment.

CONTROL AND METACONTROL

A familiar ordering of branches of science illustrates the concept of abstraction. Physics is more concrete than chemistry, according to this arrangement; chemistry more concrete than biology; biology more concrete than psychology; and psychology more concrete than sociology. In traditional work, eachof these branches dealt at bottom with vast composites of the units identified by its more concrete neighbor. The smallest biological unit was composed of a host of chemical units, and similarly at every level. The phenomena of biology emerged from those of chemistry; a biological event could be characterized as a pattern formed of a very large number of chemical events, properly distributed in time and space. Biologists without knowledge of biochemical detail could and do successfully write about biological ideas.

Yet it could be argued that this view of the sciences is sophisticated. In a more naive view, each of the sciences stands as an independent system of abstractions from the same basis of primary observations. Biology developed without reference to chemistry; the hope of establishing a relationship between life processes and chemical processes arose after both sciences had formed. By that time, each had its laws; the problem was, therefore, whether the units of biology could be characterized in chemical terms, and whether biological laws could be explained by reference to the chemical characterization of biological units and to chemical laws. Until recently, progress was slow. The simple geometrical conception of the double helix and further ideas about the interaction of physical particles as determiners of the folding of massive chemical structures are largely responsible for the rapid progress of recent years. Since folding is a physical, not a chemical, concept, it seems fair to say that direct reduction of biology to chemistry failed. Rather, biology is to be understood by using chemical and physical concepts jointly.

The origin of metagramming is a study of cognition, the process of thought (ref below), which led to a fourfold complex. The first component is a stack of perceptual levels organized to govern behavior as in the cybernetic theory of Powers (ref below); each level abstracts from the one below--the

Ref: Hays, David G. Cognitive Structures. HRAF Press, New Haven, 1981.
Powers, William T. Behavior: The Control of Perception. Aldine Publishing Co., Chicago, 1973.

horse is perceived at a lower level than the gallop. The second component is another hierarchy, orthogonal to the first; its base is the entire perceptual stack, and its effect is to organize the choices at different levels of perception to ensure their mutual compatibility and joint effectiveness. The third component is a structure of spatial and temporal relationships over the first and second. The fourth component organizes purposes, beliefs, and other modalities of thought; its base consists of the entire threefold complex beneath it. Within any one component, a higher or more abstract level works on the single level beneath it; but a component of higher degree works on the entire complex of components of lower degrees, and can therefore contain abstractions of system relationships on the broadest scale.

A familiar and striking correlate of the sequence from physical through social systems is the increase in self-control; and within social systems a further progression occurs. Physical systems tend to a uniformity that has been called heat death. Chemical systems tend to equilibrium. Biological systems metabolize and reproduce. Psychological systems seek food and mates, and tend to build secure nests. Social systems educate their young and set up diverse organizations to maintain their characteristic structures. What we take to be more abstract entities and attributes are more actively defended than concrete entities and attributes. Or, to take a different point of view, natural evolutionary processes can occur only insofar as massive organization yields new and vastly greater capacity for maintaining internal coherence.

Cognition is as natural as physiology; medically, cognition is just the physiology of the brain. Human cognition, with four degrees of hierarchical organization, is vastly more powerful than that of any other animal--it has created and maintained civilization. If the foregoing speculations about both cognition and molecular biology survive thorough testing, then one may suppose that capacity for self-regulation depends on the existence of degrees of control that have access to all prior levels and degrees. This phenomenon is distinct from generalization (elimination of detail) and abstraction (formation of a pattern in homogeneous detail); where a pattern embraces details of mixed levels of abstraction, the pattern is meta to the details. This new usage of 'meta' is at least compatible with the usage in such terms as 'metalogic' and 'metalanguage', and at best is the explication not previously formulated.

Subject to some simple and obvious criterion, generalization may be unique. Thus, to take a trivial example, nothing serves the purpose of guessing as well as the statistical mean of a distribution--given an obvious measure of the importance of error in guessing. But abstraction seems not to be unique under any simple or obvious criteria. The uniqueness of the horse's gallop in perception shows only that the nervous system has evolved over a long time in response to criteria that are not obvious and not necessarily simple. The gallop is the unique perceptually valid abstraction, but making a computer recognize it has been hard work. The physician, trained to recognize diseases --which are abstract entities--from a combination of direct observation of the patient, consideration of the case's history, and study of laboratory findings, may list several alternatives for differential diagnosis: Each alternative disease is an abstraction that fits available knowledge, and only response to treatment may suffice to identify the valid abstraction. Nor

is the encompassing abstraction of disease unique. The Christian Scientist has a different abstraction, and so does the psychosomaticist.

The recent success of molecular biology, with its physical and chemical basis, suggests that the nonobvious criteria needed to fix abstractions come from deeper levels. The claim that a distribution is normal is more strongly evidenced by exhibiting the process that generates the data than by the data themselves. If a horse is galloping, it should be moving rapidly across its background. Metastructure brings information from deeper levels to bear on the choice of abstractions at higher levels, promoting the security of internal regulation.

The study of abstraction therefore gives way to the study of abstractive systems. Such a system has three or more degrees. Each degree has units that correspond to patterns in the previous system. A pattern in the third or any further degree accords with relational structures of all prior degrees. The relational structure of particle physics consists of the forces between pairs of particles; biological patterns must accord with them.

COMPUTATION

Computation is not a science or a family of sciences, but it can have a structure of abstractive levels comparable to that of science. Artificial evolution can supply externally the maintenance that natural evolution must supply internally; but the cost is high. One virtue of a high level of abstraction in dealing with computers is the escape from attention to detail; but unless the abstraction serves at the same time to provide greater capacity for self-maintenance, the complex software must inevitably cost too much. In short, a metastructure is necessary if computers are to maintain themselves.

The lowest level of abstraction in contemporary practice is that of the chip, which provides registers and elementary operations including some primitive means of control. Further levels would then have data structures, composite operations, and sophisticated means of control. As biology needs both chemistry and physics in its basis, so a third level of computation would seem to need both the first-level chip and the second-level basic software in its basis. And a fourth level might need all three prior levels, as psychology may prove to need physics as well as chemistry and biology.

The chip creates three classes: Values, addresses, and operators; the operators calculate new values from old, move values between registers and external addresses, and control their own sequence. An address or an operator can be taken as a value; address arithmetic enables the programmer to work on strings, matrices, and other arrays of values. Logical operations on operators make sense, at least on some chips. But some values, provided by the user, are not to be taken as either addresses or operators. Operations on values that will subsequently be used as addresses or operators belong to the domain of control, and are conceptually distinct from operations on users' values. The chip deals in assignment, which is a relation between value and address.

As a second level, we can take the method of applicative programming: Control the application of functions to arguments. Concretely, the flux of values at addresses must go on; but abstractly, computation is the nested evaluation of functions. Like some of the operators on the chip, a function takes values as arguments and yields new values as results. The structure of a computation is the linkage that makes the value of one function argument to another. Control is segregated; this more abstract view is considerably easier to understand than calculation on the chip. The elimination of addresses has an enormous effect in reducing detail for the programmer.

A two-level computing system operates under the governance of a third component, meta to the chip and the language of functions, a compiler or interpreter which sees to concrete realization. Abstraction brings with it some possibilities of self maintenance. A function can be characterized by the number and types of its arguments, and by the type of its value: The integer addition function takes two integers as arguments and gives one integer as value. Coherence has new meanings in a two-level system.

For a third level, we propose the name of adjustive programming. This mode can be stated in the formula: Adjust the purposes of the system to external conditions. The computing system has its own purposes; the control structure is not external but internal. The processes of the system are given, but the internal control structure dictates that they be executed according to the content of transactions with the environment. The user is not in control, but truly in interaction with the system. Purposes must be realized by complex or simple functions, and functions in turn by operations on the chip.

The metastructure of a three-level system has an element like an interpreter or compiler to realize functions on the chip; a second element of similar kind to realize purposes as organizations of functions; and a third element to interlock the first two. Chip, functions, and purposes are of the same degree, and the interpretive component with its three elements is one degree higher. A component that supervises these two degrees jointly is then of higher degree still. Dealing with concepts of the system as a whole, the last component is uniquely capable of, for example, economizing operation in the chip in the realization of a given purpose.

CULTURAL EVOLUTION

The thousands of distinct cultures that have been studied during the last hundred years differ among themselves with respect to social and political organization, means of producing and distributing food and other material goods, and systems of belief about nature. These differences correlate among themselves statistically, convincing many anthropologists that culture has evolved. A way of putting the matter is that control over complexity grows. The present section describes cultural evolution as a sequence of cognitive jumps, each occuring as a new level of thought arises.

One might conclude that the sequence is not ended, and that another jump could take place even now.

FOUR STAGES

Some 250,000 years ago, but the dating is uncertain, a species lived on a plain in Africa; it had developed a manual-visual correlation, superior to any that had ever existed before, which supported instrumental activities of a high order. The species was using tools which directly extended the abilities of its facile hands. It had also developed a vocal-auditory correlation of superior quality, which facilitated social activities of equally high order. The species became social hunters and gatherers. After a long period, a biological change produced a correlation between the manual-visual and vocal-auditory correlations, and the new correlation supported speech. The culmination may have occurred 50,000 years ago; this whole paragraph is speculative.

Between 5,000 and 10,000 years ago, some bands of hunting and gathering people began to manipulate plants and animals in a new way. Some of these bands developed techniques of fertilization with manure and irrigation with dammed and channeled streams. Within a few thousand years, they began to keep visually recognizable records of inventories, and then of events; they had made a new correlation between the manual-visual and vocal-auditory channels, and it supported writing. There is no reason to suppose that this new correlation came as the result of a biological change. Some say that the invention of writing fulfilled a new need, as if humanity were and always had been omnipotent, able to meet its needs on all occasions. A more plausible view is that the management of plant and animal life created a new environment, one not comprehensible with earlier methods of thought, and that reflection on the new kinds of events led to a new, more abstract system of thought in which these new events could be understood. Late in this period, engines came into use; their framework of orientation was distinct from that of the user--devices to lift water or to break down city gates. The culmination of the invention of writing came with the appearance of philosophy, an extensive and socially recognized analysis of thought itself. (If the appearance of speech had a comparable effect, there is of course no record of it, but when did consciousness first appear?)

Between 400 and 800 years ago, some small states began to conduct commerce, manipulating the ownership and location of things in a new way. The situation had, apparently, two novel qualities. This was probably the earliest occasion when any sizable sociopolitical unit could not say, "We make what we need to support life." And it was probably the earliest occasion when inherently worthless tokens, written letters of credit or the like, were used on a large scale. Within a few hundred years, a new system of calculation appeared in Europe: The eastern place-notation for number writing, with the accompanying algorithms for addition, subtraction, multiplication, and division replaced the method of counters on boards. The introduction of calculation had its culmination in the appearance of science. Machines with internal timing, notably clocks, were invented.

Between 150 and 200 years ago, selfpowered machinery appeared in Europe, changing the forms of things (shapes, colors, combinations) in unheard of ways. Among the most dramatic of the new machines was the locomotive, which violated a rule that had stood since its formulation in ancient Greece: What moves of its own accord is alive. Again invention yielded an environment that could not be comprehended with older methods of thought, and again new methods of thought arose: Logic and mathematics developed rapidly. The sequence of external supports for cognition--speech, writing, calculation-- and the sequence of external supports for manipulation--tool, engine, machine -- come together with the invention of the computer, a machine which contains its own organ of control. The culmination of this innovation is a new system of thought, different from the science of earlier centuries.

Thus history reports four inventions: Social hunting, agriculture, commerce, and machinery. And history reports that the last two, certainly, delivered phenomena that upset systems of thought; the speculation that each of the four was intellectually upsetting seems justified. And history reports four systems of external symbolization: Speech, writing, calculation, and computation. Each followed one of the upsetting inventions. The last three had remarkable culminations: Philosophy, science, and contemporary thought. And if consciousness was the culmination of the first, that is most remarkable of all.

The time intervals are hard to measure with precision, but surely the great decreases are unmistakable. From hundreds of thousands of years down to hundreds of years is an enormous reduction. One might expect the next major change to take place within decades after the appearance of the computer, and therefore to be in progress now. Metagramming may be part of such a change.

LEVELS OF THOUGHT

One may speculate that the four levels of culture associated with speech, writing, calculation, and computation are characterized by systems of thought of increasing power. The highest rank of cognition at each cultural level is meta to the preceding system.

A person in the most primitive kind of culture lives by lore. Lore is effective but, lacking systematic organization, it is inflexible.

With the appearance of a metasystem, the person's thought is at the level of technology. The typical support of the technologist is the table, or list of facts. This is the level of writing, of agriculture.

The system of thought characteristic of the third level is scientific. The typical support of the scientist is the law; from it, the scientist can generate tables, whereas the notion of reducing a table to a law goes beyond the conceptual framework of technology. This is the level of calculation and commerce.

With another metasystem, the level of thought goes beyond the scientific; at this stage, one can consider different ways of reducing tables of facts to general laws, and one comes to realize, as Karl Popper did, that a law obtained by an arbitrary method of reduction cannot be construed as ineluctably true--a different method of data reduction would give a different law, and a different law would constitute a different reality. Such is the system of thought with which we live; it gives us great power, but since it does not give us a unique reality it cannot give us security.

It may be that with passage to another level of thought, security comes back. It cannot be identical with the old security. The scientist of, say, 1850 stood secure in the knowledge that a table of observed facts could be reduced to a law in one way, and, since he knew exactly one way to make the reduction, in only one way. His security resided in his own ignorance. But we know that data reduction can be accomplished in many ways, leading to different realities. What we require is a rule of choice among manners of reduction, one that is more attractive than any particular reality.

DEVELOPMENT OF PROGRAMMING METHODS

Metagram's obvious but difficult proposal is to make the computer help more in software development. The advance lies not in this obvious suggestion, but in a shift of perspective. Automatic programming is at present considered a topic for research, and not a particularly hopeful topic. A new application calls for a new program, and writing a computer program is a creative task. The familiar and horrifying image of monkeys typing until Hamlet appears by accident; the nasty image of shaking the pieces of a jigsaw puzzle in a large box until, at random, they fall together into their own solution; these and other hopeless notions come to mind when one suggests handing over a creative task to a machine. But they come to mind only for as long as one sticks to a familiar perspective. From a new point of view those images disappear, as paradoxes have disappeared in the past. As before, the creative task turns out to have routine aspects that a machine can perform. A historical sequence may help to make clear not only what the new perspective is but also why the old perspective has held the art static for so long.

LISTS OF INSTRUCTIONS

Superficially, primitive computer programs consist of "long lists of instructions"--language that BUSINESS WEEK employed in 1980 to characterize the contemporary scene. In a deeper analysis, the first genuine computer program was more than that; even a primitive kind of program must jump. As a text, the program is a list of instructions. As a computation, the program is manifest in a sequence of executions. But as a program, the program has loops and branches. A loop in a program text may execute over and over again; the loop ends with an instruction which is a conditional jump back to the beginning of the repetitive part. If the jump is not conditional, the loop executes again and again forever; with a conditional jump, execution repeats until the condition is satisfied--until the approximation is good enough, or until the input data are exhausted, for example. A branch is a part of a program text that executes only when a specified condition arises.

Programming with loops and branches is inherent in the fetch-execute cycle that defines computation. The mode of operation of the standard computer is fixed by this standard scheme:

> Execute the current instruction. (This instruction is stored in a special register connected permanently to the built-in operations of the computer.)
>
> If the current instruction is a (conditional) jump, then reset the instruction counter (if the condition is satisfied); otherwise, advance the instruction counter one step. (The instruction counter is another special register that always contains a pointer into machine memory. The reset value is stored with the jump instruction.)

Fetch the next instruction. (The content of the memory location specified by the instruction counter becomes the current instruction.)

Repeat.

It is hardly an exaggeration to say that every computer is executing this loop whenever power is supplied. The invention of the fetch-execute cycle made it possible to work through indefinitely large files of data with small programs; the cycle binds the temporal sequence of execution to the spatial store of instructions in an absolutely novel way.

What a computer achieves by executing instructions is systematic change in the contents of storage cells. Taking the contents to represent numbers, words of a text, or other kinds of data, the instructions calculate arithmetic or other functions and store the results. In formal terms, one takes the storage cell as a variable and assigns a value to it. Unlike mathematical variables, however, the storage cells of the computer have numeric addresses. At the fundamental level, the orderly selection of storage cells depends on address arithmetic. To add up numbers stored in consecutive cells, the machine alternately adds contents and increments the storage address. To keep track of data in storage, the address of one cell must often be stored as the content of another. The numeric indexing of storage is a major invention, helping to fix the perspective of the computing art.

So the art of programming begins with a clear and definite concept of the program, derived from the fetch-execute cycle and numeric addressing: A list of instructions to be executed in order, resulting in value assignments, with now and again a jump forward or backward in the list, all organized around the arithmetic of storage addresses.

MACROS AND SUBROUTINES

The next grade of programming recognizes that portions of the code, fragments of the list of instructions, occur again and again. Two methods come into use, saving the programmer the necessity of writing out the same fragment again and again. One method is the macro, one the subroutine. With either method, the text as written cannot be submitted to the bare computer for execution; instead, another program called an assembler must work on it. In the original text the programmer writes the name of a macro or subroutine as often as necessary; the full text appears just once somewhere. A macro assembler replaces the name with the full text in every occurrence, yielding a longer text for the complete program. As extended, the text is ready for execution by the computer directly. With subroutines, on the other hand, the text that the programmer writes is not lengthened (or not much) by the assembler. The name of a subroutine is accompanied each time it appears in the program text by a word that changes the mode of operation of the machine; the word is usually CALL. To execute a subroutine call, the machine interrupts its flow of work, executes the instructions in the text of the subroutine, and finally returns to the point of interruption. The method of subroutine calling keeps the program smaller, and is much more widely used.

Ordinarily, a macro or subroutine needs arguments--numbers, say, on which to operate--and ordinarily but not always a macro or subroutine yields a result and must end with an assignment. The arguments are the contents of certain storage cells, and the assignment is also to a cell, known to the computer by their numeric addresses. Since addresses are less easy to remember than names, assemblers perform a translation for the programmer: The original program text can contain names for variables. Various syntactic forms are provided to put together the name of a macro or subroutine and the names of its arguments and, possibly, the variable to which its result is assigned.

Named variables are convenient, but programmers often treat collections of data and would find the naming of each individual datum quite inconvenient. Address arithmetic reappears in subscripts; thus X(3) may be the third member of a collection X. And variables appear in subscripts: X(I) is found by taking the value of I as index into the collection X.

Programming with conditional jumps is hard enough to warrant invention of other devices. One is a looping formula, such as

```
for i := 3 step 1 until 10 do [task] loop
```

To assemble or, more properly, to compile this formula, the system reserves a storage cell for the dummy variable i. The initial value 3, the increment 1, and the final value 10 are also kept for use during computation. The code is a loop; part of the code carries out the task, and part increments the dummy variable, making a conditional jump back to the beginning of the loop until the final value is reached or passed. The dummy variable can appear in the task program, perhaps as subscript; to form a sum, one might write

```
for i := 3 step 1 until 10 do Sum := Sum + X(i) loop
```

At each step the value of i serves to choose a storage cell relative to the origin specified by X; the content of that new cell is added to the content of the cell known as Sum, and the partial total stored there again.

When the system compiles a looping formula, it initializes the dummy variable; that is, it provides code in advance of the loop that stores the initial value (3, in the illustration) in the cell reserved for the dummy. But the programmer must write explicitly other initializations; in the example, Sum must be set to 0 immediately before each execution of the loop.

Operations on programs were foreseen from the beginning. A computer has only one kind of storage, in the sense that data and programs can be kept interchangeably anywhere. An operation can be carried out on the name of a person or, equally well, on the name of an operation. Arithmetic can be done on income data, or on addresses of storage cells. Operations on macros could adapt them, as they are copied into different parts of a program, to suit their various contexts. In practice, automatic adaptation of macros is unusual, and the more widely used subroutine cannot be adapted. The methods that have been used by most programmers strictly separate operations on programs from those on data: First assembly; then execution of the assembled code. As

the art of computation advanced, the habit of segregating and restricting operations on programs came to be almost universal.

FUNCTIONS AND THREADED CODE

A strictly functional mode of programming eliminates all reference to time and place of computation. Neither the sequence of execution nor the place in storage where arguments and results are kept appears in the program text, even through names of variables to be translated into addresses. A function has a value; the syntax of a functional language must let the programmer show how the value of one function becomes the argument of another. The ordinary algebraic notation with parentheses can serve:

```
Print ( Square ( Average ( Input Input ) ) )
```

Here each word names a function. Perhaps Input is a function that obtains a value from an operator at a keyboard; Average and Square perform arithmetic; and Print shows the result on a screen.

The example contains no variables, no assignments, and no indication of sequence; by contrast, the same program written in a system of lower grade might read as follows:

```
A = Input
B = Input
A = Average ( A, B )
A = Square ( A )
Print ( A )
```

Here is a list of instructions, and all but the last are assignments. After A is used for the average and then for the square, it no longer contains input; if the programmer forgets this fact, and consults A subsequently, error will result. The functional version no longer has the input, possibly to the programmer's regret; but it avoids the illusion of having it.

Functions are defined in two ways. A primitive function is defined by a list of machine instructions; when the function is needed, the machine executes the instructions by way of the fetch-execute cycle. A nonprimitive function is defined by a syntactic combination of other functions, primitive or not; when it is needed, the functions in the definition must be executed in the proper order and with the proper arguments. Since a definition can contain nonprimitive functions, the execution is recursive; the expansion of a definition may have to begin in the middle of the expansion of a larger one.

A function dictionary can be stored in the form of threaded code. With this structure, a simple executive can link the level of functions with the level of machine operation. To keep the executive simple, however, two prerequisites must be met: The sequence of operation within a definition must be normalized, and the names of functions must be reduced to numeric addresses.

No function can be computed until its arguments are available. In the example above, the translation into a lower-grade language shows the necessary order. Many current compilers make such translations on program texts; current threaded-code languages require the programmer to write parenthesis-free text:

Input Input Average Square Print

Executing this code from left to right is possible, since the arguments of each function are available when wanted. A functional system reserves space for the intermediate results (in a stack), so the programmer needs no variables.

If the definitions of functions are stored in the dictionary as strings or other syntactic combinations of words, the executive must search for their definitions in turn. But when a definition goes into the dictionary it goes to a specific location in machine storage, one with a numeric address. If this address represents the defined function thereafter, search during execution is avoided.

The executive of a threaded-code system does this:

> For each word in a definition,
>
> If the word has a primitive definition, apply the fetch-execute cycle; otherwise, apply the executive.

This simple definition is comparable in importance to the fetch-execute cycle; it raises the conceptual level of computational structure from iteration to recursion, with the concomitant benefits in clarity and simplicity. Like other recursive processes, this executive requires a stack. Given a definition of highest order to process, the executive must go through it word by word; it needs a pointer to keep track of its place in the definition. When the executive encounters a word with nonprimitive definition, it must retain its present pointer and create a new one to move through the embedded composite. A stack keeps all of the pointers in order from current through highest order. At the end of a definition, the current pointer disappears and the next takes its place. When the stack is empty, the outermost definition is done.

Around 1970, John W. Carr III and students at the University of Pennsylvania created a system that they called "the growing machine" (refs below); Charles Moore later developed and marketed FORTH, and versions are now available from several companies. The invention of the threaded-code dictionary of functions came earlier than the vogue of structured programming; yet it achieves directly and fundamentally what the structured programming idea imposes superficially. And the growing machine provides economy both in

---

Ref: Bair, Robert Paul. The Use of Multiple Associative Memories in Programming the Growing Machine. MA Thesis, U. Pennsylvania, 1968.
Albertson, Mark D., and John W. Carr III. GROMAC-71: An Interpretive System for Development of Special Purpose Languages.

the size of the program and in speed of operation. Informal reports suggest a substantial saving of the programmer's time.

The machine grows as the user adds new definitions to the dictionary. To add a definition is relatively easy for the user, and the system that makes it easy is relatively simple. Working in this mode, users tend to write many short definitions rather than a few long programs. A "long list of instructions" has a complex syntactic structure; to understand a long program is a difficult task for a human being. A large dictionary of functions has at every point a simple syntactic structure; the burden on the user is to remember the vocabulary.

The chess master seems to know a large vocabulary of board patterns, as the work of Herbert A. Simon and associates has revealed. Medical students learn thousands of terms. Many sophisticated arts employ large vocabularies. (Ref below: Expertise involves "large numbers of patterns".) Programming stands in sharp contrast by its insistence on syntactic complexity. The concept of the growing machine may facilitate the growth of expertise in this art.

A NEW PERSPECTIVE

Functions are more abstract than the instructions of a fetch-execute computer. To make functional programming work, someone had to invent a way of implementing the abstraction in machine code. McCarthy's LISP and Carr's growing machine are outstanding implementations. The next problem that arises is the design and implementation of systems more abstract than functions.

Higher-level abstractions are the missing element in natural-language programming. The way we think is the product of a long biological and cultural evolution, marked by upward steps in abstractive capacity. Natural languages reflect and embody methods of thought; heretofore, natural-language programming has adopted vocabulary and grammar, but only for concepts of the most concrete levels--the levels of fetch-execute and of functions.

Higher-level abstractions are also the missing element in automatic programming. Human creativity requires random search, but under the control of highly abstract criteria of consistency, coherence, and validity. Artificial intelligence uses concepts at the functional level; with higher abstractions, greater effectiveness could be expected.

Higher-level abstractions serve further to regulate or control concrete systems. The computer is our best example of a mechanical device conceived only when the systems of thought of the culture reached a very high level. It is to be expected that better methods for all computer users, including the most naive, can be provided within a framework specified by high abstractions.

Ref: Larkin, Jill; McDermott, John; Simon, Dorothea P., and Simon, Herbert A. Expert and Novice Performance in Solving Physics Problems. SCIENCE 208: 1335-1342, 20 June 1980.

Finally, higher abstractions supply the natural mode for fusion of matters of incompatible types. Functions and data structures are incompatible in this sense; a higher abstraction can fuse them. Computation and the user's operations are incompatible; the programmer's task is to make them fit. The fit is apparent only with abstractions higher than those implemented thus far; the programmer acquires these higher abstractions through apprenticeship.

One obvious approach to the problem of abstraction is to take each step separately. The threaded-code executive implements functions in hardware. Imagine another level of abstraction and write its executive as a scheme of functions. Imagine other levels, and for each make the implementation using the most abstract system available.

Another, equally obvious approach is to design a universal executive. With it, any imaginable level of abstraction could be added to an existing system by inserting details. Abstraction promotes diversity; a universal executive would facilitate the examination of the many imaginable alternatives. This approach seems to be the one found in nature: Human cognition might best be described as a universal executive for systems of abstraction, and human culture history as an exploration of possible systems. This approach is adopted in metagramming.

A universal executive must make a three-way distinction among values, structure, and mode. These three levels of information correspond fairly well with data, indexing, *and* control. A value might be a number or a textual character; a structure might be a matrix or list; and a mode might be actual or potential. In computation, all executives have to do with actualization of potentials; metagramming adopts an explicit treatment of mode as its highest method of control.

Every executive must provide for the calculation of functions, giving each function its argument and keeping track of the result it returns. The basic functional form in metagramming has four slots for these purposes:

> Information is a slot for the argument
> Function is a slot for the calculation on an element
> Registration is a slot for the result
> Application is a slot for the mode of combination of
> current and previous results

In each slot, provision must be made for the levels of value, structure, and mode.

On the value level, the Information passes to the Function as argument; the resulting value and the Registration pass to the Application as a pair of arguments; and the result goes back to Registration to end a cycle.

On the structure level, an indexing process occurs in each slot during each cycle. If the filler of the Information slot is a collection of values, then indexing moves through the collection over a sequence of cycles. Or the Registration slot may be planned so that after many cycles it will contain a collection of values; the indexing process builds the collection.

On the mode level, if a slot is filled only potentially then it must be actualized before the indexing process takes place. What is stored outside the fast memory of a computer is potential in a certain sense and is actualized by an input process. Recursive computation can be said to involve potential terms; the recursive calling actualizes them. An incompletely defined function is potential; automatic programming would actualize it.

The metagramming executive is universal in the sense that each slot can be filled, on the levels of structure and mode, with either primitive or nonprimitive functions. (In the ordinary threaded-code executive, machine instructions fix structure and mode.) A nonprimitive function is specified by filling the slots of the functional form; such an object can have a name, and the name can fill a slot in another form. The aim is to give the designer direct access to every level and aspect of computation, with the expectation of stimulating work at higher and higher levels of abstraction.

The executive is no more universal than the fetch-execute machine in the sense of absolute power; the fetch-execute cycle can accomplish any feasible computation. Absolute power and effective power are not the same, however; effectively, computation is the work of persons and machines jointly. Metagramming may provide enough points of control to let the designer set up an effective system for each of many different applications.

APPLICATIVE THEORY OF COMPUTATION

Information-processing machines are not all alike from the point of view of the user. The universality of the Turing machine, which is built into every commercial stored-program computer, guarantees that every machine can do every computation; what one can do, all can do. But universality does not guarantee ease of use. The Turing machine itself makes the proof of universality easy, but makes any computation hard. From the user's point of view, Turing's design is awful, its universal power unusable. Commercial machines are not more universal than Turing's, but they are easier to use. In fact, they are so easy to use that millions of them are in actual use today; but not so convenient as to prevent the growth of a software problem. But metagramming requires that machines become users of other machines, and a machine that a machine can use must be very, very easy to use.

A pivotal element in the development of metagramming is an ordered series of machines, each much easier to use than its predecessor. The series extends one step beyond the contemporary machine, and that extra step brings the art to a kind of machine that even a machine can use. If the series can be carried one step further, the next kind of machine should be easier still; but the next step is not obvious, and one cannot forecast its discovery.

The series of successively easier-to-use machines is defined by the number of kinds of slots the machine requires.

ONE-SLOT MACHINES

The typewriter is an information machine, albeit a simple one. Its simplicity is helpful in the introduction of notation and of the idea of slot. A typewriter has a keyboard and a set of type bars. When the user strikes a key, the corresponding typebar strikes the paper, leaving a mark. The only information resides in the user's choice of a key. Thus

```
Typewriter ( Information )
```

The name of the machine is followed by a pair of parentheses. Inside the parentheses stands the name of the one and only slot. The word Information holds a place which can be filled in one or more ways. But what the machine does with the filler of the slot is always the same: It prints the corresponding mark on paper.

A typewriter with an external switch marked LOCAL-REMOTE gives the user two ways of filling the slot:

```
Typewriter ( LOCAL )
Typewriter ( REMOTE )
```

In the LOCAL position, the switch binds the typewriter to its own, physically

attached keyboard. In the REMOTE position, the switch binds the typewriter to some other keyboard or to a computer. The switch is a machine that operates on the basic machine.

Another way to think of the machine is as a physical device, which the typewriter certainly is; only consider the printing part to be separate from the keyboard part. The printer has a socket, labeled Information. The user plugs the keyboard into the socket. The expression "Typewriter (Information)" is a functional form; the word "Information" names only the slot. With the slot filled, say by "LOCAL", the expression becomes a function.

The typewriter is easy to use for its simple purpose. Other one-slot machines have theoretical and practical interest, but an inventory and discussion is not necessary here. And a one-slot machine to serve the purposes of the general-purpose computer would be so hard to use, or even to explain, as to hold up the main line of exposition. It would be a Turing machine.

TWO-SLOT MACHINES

The adding machine has a memory as well as a keyboard. Machine users and builders have the term "register", which supplies a name for the second slot:

Addingmachine (Information Registration)

Again, the name of the machine is outside the parentheses and the words inside are place holders. In the typical mechanical adding machine, the slots are filled at the factory:

Addingmachine (KEYBOARD WINDOW)

The user presses keys to enter new information, and the new sum appears in a window.

And, again, what the machine does with each slot is always the same. The adding machine forms the sum of the Information number and the Registration number; since addition is symmetrical, it treats the two slots alike up to this point. But the machine puts the sum into Registration, not into Information; and this distinguishes the two slots: Registration is the slot into which the machine puts its results.

The practical adding machine has, besides the numbers on the KEYBOARD, another external control; with a lever or a button, the user can force 0s into every position in the WINDOW. This is an initialization, and like binding it is an operation performed on the basic machine. Binding converts a functional form into a function by putting the names of objects into slots; initialization fixes the values of the objects: WINDOW is an object, and 0 is a value.

THREE-SLOT MACHINES

The contemporary computer is probably best described as a machine with three slots; more precisely, with three kinds of slots:

Computer (Function Information Registration)

Having an Information slot, the computer can accept input. Having a Registration slot, the computer can retain the results of past computations. And having a Function slot, the computer can do some informatic work on the input before combining the new information with the old. Giving the slots the names familiar in contemporary computation, Registration is like the Accumulator, Function is like the Instruction Register, and Information is like the cell where an argument is placed for a program to use.

Binding (like the typewriter's REMOTE-LOCAL switch) and initialization (like the adding machine's reset to 0 total) must be performed inside the computer. The contemporary computer operates millions or billions of times faster than human fingers; it would be folly to wait for the human operator to set the accumulator to 0, bind the several slots, and start the computer every time. Of course, it would be possible. The operator could type

ADD (SIN X-ARRAY ACCUMULATOR)

and so start a machine that would construct the sum of the sines of all the numbers in the array X, leaving the total in the accumulator. A pocket-size machine is used in roughly this fashion. But a machine designed in this way wastes most of its time waiting for the typist.

Instead, the computer binds and initializes itself, according to instructions in a program; and, of course, some of its programs instruct it to load and execute new programs.

However, binding and initialization are not well differentiated in the contemporary machine, and quite a few special tricks are used in the absence of a general scheme of principles. One trick, familiar to everyone, is the automatic advance of the program counter. The user must see to it that a "program" is stored sequentially in machine memory. The program counter then moves line by line through the program, fetching a new filler for Function and a new filler for Information from each line and executing the specified Function on the specified Information. The name for this trick is "Fetch-Execute Cycle", and it is used everywhere. Another trick, also used in all computers, is called "Conditional Transfer". Some filler of the Function slot causes the corresponding Information to be used for rebinding (or for re-initialization; it is hard to tell one from the other in a computer) and thus to effect a "Jump".

The computer is a complex machine. It manages to combine great power with relative simplicity of use by managing, with its three-slot form, to control its own binding and initialization, that is, control of is own operation. But its three-slot form is not adequate for conceptual distinction among binding, initialization, and computation of functions. Another slot is needed.

FOUR-SLOT MACHINES

A machine with four slots is a new kind of informatic device. It deserves a new name; it is not a computer. Instead, Metagram proposes to call it a Controller.

Controller (Application Function Information Registration)

It has the three slots of the Computer, and a fourth named for the style of programming identified in the current literature as appplicative. David S. Wise describes "Applicative Programming" as "A style of programming that controls the application of functions to arguments".

What a Controller does is this: Put into Registration the value obtained by executing the Application with two arguments, the first of which is the prior value of Registration and the second of which is the value obtained by executing the Function with the value of Information as argument. More compactly,

R <-- A (R, F (I))

where the arrow is an assignment and A, F, I, and R are (values from) the four slots.

Summation is an elementary example. Define

Sigma = Controller (Add Identity X Sum)

So the Controller stores as a new Sum what it gets by adding the value of X (the Identity function does not change it) to the prior Sum. Repeated, this operation would construct the sum of a collection of Xs; below, the method of repetition appears.

Some other examples: To sum the squares, change Identity to Square. To make a list of the squares, change Sum to List and change Add to Attach:

ListSquare = Controller (Attach Square X List)

What happens is

List <-- Attach (List, Square (X))

the prior list grows longer with the attachment of another square to it.

Like the computer, the controller must bind and initialize itself. With elaborated versions of the four kinds of slots, introduced further on, clarity of distinction will be achieved.

TYPES OF CONTROLLERS

A more compact notation is convenient:

G (A F I R)

where

G is a Controller with four slots
A is an Application
F is a Function
I is an Information
R is a Registration

Another type of Controller is

T (A F1 I1 F2 I2 R)

with six slots: Two each of the Function and the Information types. Not only does T have six slots, but also it has the data flow

R <-- A (R, F1(I1), F2(I2))

The first Information slot supplies an argument to the first Function slot, and the second to the second.

As a rather elegant example of T, define TRUTH by

T (MATCH INTERPRETATION STATEMENT PERCEPTION SITUATION BELIEF)

Tarski put it that "'Snow is white' is true if and only if snow is white." The TRUTH machine gives

BELIEF <-- MATCH (BELIEF, INTERPRETATION(STATEMENT), PERCEPTION(SITUATION))

In other words, a truth value BELIEF results from testing the correspondence MATCH between understanding (by INTERPRETATION of what is said, a STATEMENT) with the comparable understanding (by PERCEPTION of what is available to the senses, a SITUATION). But the TRUTH definition also takes into account prior BELIEF.

Another example of T is the validation of programming languages; and it is very like the first. Define VALIDATE by

T (MATCH P-ANALYSIS PROGRAM A-ANALYSIS FORMULA VALIDITY)

or, in data flow,

VALIDITY <-- MATCH (VALIDITY, P-ANALYSIS(PROGRAM), A-ANALYSIS(FORMULA))

Take an algebraic expression FORMULA and apply algebraic methods to it by A-ANALYSIS. Take an expression, a PROGRAM in Basic for example, and apply

programming methods to it by P-ANALYSIS. MATCH the results, subject to the prior value of VALIDITY.

The examples in the present section are no more than indicators that the functional forms, that is to say the lists of slots, called G, T, and so on are good enough to give the right outlines for familiar problems. With a simpler form than T, be it noted, one could check the validity of a program; but only by giving priority to either the program or the algebraic formulation of the same problem. And taking the point of view of either program or formula makes the task of comparison more difficult.

G is a Controller with four slots. T is a Controller with six slots. Thirdly, a machine with eight slots is the metagramming Controller

M (A F1 I1 F2 I2 F3 I3 R)

Each of the three Information slots supplies an argument to the filler of the corresponding Function slot. The data flow is

R <-- A (R, F1(I1), F2(I2), F3(I3))

In a trivial example, take F1 as Identity, F2 as Square, and F3 as Cube; take each of I1, I2, and I3 as Integer; take R as Table, and A as Attach. Then one step extends the Table by adding an integer, its square, and its cube. Even this example has the interest of showing that the slots can be interlocked; by putting Integer in several slots at once, the example makes the table come out as it should. This interlocking, or diagonalization, is needed in crucial places hereafter.

CONTROL: INDEXING AND DECISION

Computation is more than calculation because it chooses what calculation to do. This is the problem of control in a broad reading of the term, but with metagramming it is convenient and helpful to make distinctions within the broad category. Three levels emerge: Calculation, indexing, and decision; they operate on values, structures, and modes (pp. 24-25). At the first two levels, metagramming provides a convenient way of organizing familiar work. At the decision level, new convenience becomes significant.

INDEXING IN STRUCTURES

In algebraic terminology, one extends a calculation from a single datum to a collection. Summation, for example, must take its addends one by one until is has formed the sum of them all; it needs extension through the collection of addends. This is a small example; but the functional form G has several slots, and another computation may require the Application, the Function, and the Information all to be collections. These collections may be organized as data structures.

In an extension, three kinds of information are necessary: Current value, a pointer to the following value, and a specification of the kind of organization or structure. To cite some examples of kinds of structure,

CONSTANT	A single number, unchanging
SUCCESSOR	A number, increasing by 1
VECTOR	A sequence of numbers
MATRIX	A double sequence of numbers
LIST	Entries stored arbitrarily, linked by pointers
TREE	Entries stored arbitrarily, with lists of pointers
FOREST	A list of trees

The necessary information can be represented in an expanded version of G:

```
G ( A0  F0  I0  R0          Values
    A1  F1  I1  R1          Pointers
    A1' F1' I1' R1' )       Structure types
```

Each of the four kinds of slots is expanded into a triple. A single calculation uses the values; extension needs pointers and structure types.

The 0-level form G0 (A0 F0 I0 R0) is the functional form of a single operation; a simple machine. The composite G, with slots of both levels, is the form of an extended function.

A structure type such as LIST or TREE corresponds by intent with a manner of extension; hence it can serve as the name of a primitive operation in the metagramming system. Suppose, for example, that the filler of the in-

formation slot in G is a list; the filler of I1' is LIST. When LIST is executed, it uses I1 as a memory pointer to a pair of cells, and moves their content into I0 and I1.

Each slot has a structure type, and indexing is synchronized in the four slots. Consider POWERSIGMA, which sums the i-th powers of the entries in its sequence of arguments: It adds the first term, the square of the second, the cube of the third, and so on. For illustration, make the arguments the consecutive integers provided by SUCCESSOR.

```
POWERSIGMA = G ( ADD    IDENT   1       0
                 -      Pointer -       -
                 CONST  LIST    SUCC    VARIABLE )
```

The values shown on the first line are the initial values, prior to any calculation. The dashes on the second line appear because CONST (constant) and SUCC (successor) need no pointers. The word Pointer represents a memory address; tne function is a list of functions, executed in turn.

To recapitulate: The execution of A1', F1', and I1' sets current values A0, F0, and I0. The execution of F0 and then A0, with I0 and R0 available as arguments, yields a new value. To complete a step in the extension, execution of R1' registers this value; VARIABLE makes a replacement, ATTACH lengthens a list, and so on.

DECISION BY MODE

Real computation extends basic operations through finite structures; the computation must cease when it exhausts the data. This is one problem of control. Another is that a criterion for success may be part of the specification of the task; a square root computation ends when the square of the approximate root is close enough to the original number. But the more subtle problems of control are of a different kind.

To this point, bindings of functional forms have been presented as if the fillers existed in explicit form: Files ready for the I slot, functions fully defined and ready for the F slot, and so on. Giving up the assumption of readiness entails adoption of a more powerful control method, but frees the designer of the work involved in validating the assumption. If the metagram system can cope with inexplicit fillers, the programmer does not have to ensure explicitness. Extension was a 1-level operation; coping with inexplicit fillers is a 2-level operation, and covers the simpler problems of control.

For another level of operation, the functional form must be expanded:

```
G (     A0      F0      I0      R0              Values
        A0'     F0'     I0'     R0'             Value types
        A1      F1      I1      R1              Pointers
        A1'     F1'     I1'     R1'             Structure types
        A2      F2      I2      R2              Categories
        A2'     F2'     I2'     R2'     )       Control types
```

Some value types are 2-byte integer, 4-byte integer, and character. Categories include primitive function, homogeneous structure, and others. Among control types are Actual and Invokable.

In general, the operation of a filler at 0-level or 1-level yields a status indication as an argument to the corresponding filler at 2-level. For a simple illustration, take the I vector to represent an explicit list. I2' is Actual, so execution of I1' is appropriate. But I1' is LIST, and uses the pointer I1 to pull the next entry into I0. If I1 is NIL, then no next entry exists; the failure of I1' is reported to the control filler in I2', which terminates the operation of G; whatever stands in R is the end result of the computation.

Or take the problem of approximation. Take the value I0 to be a number for which the square root is required. The value R0 is an approximation. The category F2 is identity; all operation in this slot is therefore skipped. The application A0 is a function of two arguments (supplied by I and R) with results on two levels: A new approximation, and a signal that the approximation is or is not good enough. On receiving the "good enough" signal, A2' terminates the operation of G after the final approximation has been stored in R0.

Specification of a function to adapt it to its context of use shows more clearly the power of metagramming. Let the value slot F0 contain some identifier of a table of procedures, each with condition of use. The control type F2' is, say, Unspecified. The conditions of use are the value types of arguments; the procedure for Unspecified function replaces F0 with F(I0'). In other words, it uses the value type I0' to select in the table of procedures. The replacement affects the whole column:

F0	replace the identifier of the table of procedures with the identifier of a specific procedure
F1, F1'	replace this characterization of a structure containing tables with a similar characterization of the structure of a specific procedure
F2	insert Primitive, Invokable, etc.
F2'	replace Unspecified with Specified

Once the function is specified, it remains so throughout an invocation of G.

A function can be specified with respect to the value type of its argument and also with respect to the value type that it delivers; the square of a one-byte integer is, in general, a two-byte integer. Certain identities necessarily obtain in the functional form.

I0' is the type of I0 and also of the argument of F0.
F0' is the type of the value calculated by F0 and also of the second argument of A0.
R0' is the type of R0, of the first argument of A0, and also of the value calculated by A0.
A0' is identical with R0'.

If F is Unspecified, F0' may nevertheless be given by a prior specification of A; if this condition obtains, then the specification of F can be determined by both I0' and F0'. Likewise, if A is unspecified, A0', F0', and R0' can jointly serve as specifiers of the required application.

The main control structure of a powerful system must allow the slots of a form such as G to be filled with identifiers of similar forms. Certainly A and F must be so filled; but so can other slots. Processes of 2-level deal with this nesting.

Suppose a functional form has been filled, and F2' contains Invokable; a prototype for the embedded function exists, but linkage must be created. Invocation in metagramming begins by transcribing the prototype. The F0 slot contains an identifier of the prototype; invocation substitutes an identifier of the transcription. Since the argument of F0 in the invoking form is I0, that value must be copied into the transcription of the invoked form. The value type I0' of the invoking form becomes the structure type I1' of the invoked form, since it might be LIST or TREE: The invoking form may be processing a structure which is a list of trees, a tree of lists, and so on.

The invoked function can be used repeatedly if the information structure is homogeneous--that is, if the value type I0' does not change; on each use I0 and I1 must be copied from the invoking into the invoked form, but I1' and I0' remain unchanged in the invoked form. In this fashion the invoking function can process a list of numbers, using the invoked function to approximate the square root of each. However, if the information structure is heterogeneous (I0' varies in the invoking form), more work is necessary. Invocation must then be followed by specification; the invoked F may be Unspecified. Indeed, only an unspecified function could survive a change in the value type of its argument, and at the cost of respecification. On using an invoked function, tests are required: Is the Information of the invoking form heterogeneous? If so, does the value type I0' in the invoking function match the structure type I1' in the invoked form? If not, reinvoke; take the identification of the unspecified, invokable function from the invoking slot and proceed as for the original invocation. To get these tests made, the control type of an invoked function must be different from that of a primitive.

The information source that is to fill the I slot need not be in the most accessible kind of storage when a function is invoked. Suppose that the value I0 of an invoking function is an identifier of an external source; the value type I0' is, say, External. For the invoking function the control type I2' is Actual--the identifier is in the slot--but for the invoked function I2' must be different: Library-access or the like. During invocation, the system can examine the filler of I0' and, in the External case, fill the slots of the invoked form accordingly. Execution of the invoked form includes a stage in which the external information is found and made accessible; the system may consult a directory, issue a message to an operator, verify the label of a disc or tape, and so on. Naturally, such processes occur in current work, but they are often guided by an operating system and described in a language other than the programming language. The 2-level processes of metagramming encompass at least some of the activities of an operating system.

RECURSION

If a word appears in its own definition, the definition is recursive. In some systems of programming, and certainly in metagramming, recursive procedures are tolerated. A standard example is the factorial function, 1! = 1, 2! = 2 x 1, 3! = 3 x 2 x 1, ..., or

```
n! = if n = 0 then 1
             else n x (n-1)!
```

But this definition can be recast as a tail recursion, making the work iterative:

```
Factorial = G ( Product Identity Pred-n Unity )
```

The application Product goes with the multiplicative nature of the factorial. The registration Unity is initially 1, and holds successive products. The information Pred-n is a structure that gives n as first value and thereafter values reduced by 1 each time until the value would be 0 and it returns NIL. Once this function is invoked, it simply runs to completion, leaving the factorial of n in registration; no recursion takes place. Note that if n is originally 0, then no calculation occurs; since I2' gets a failure return from I1' immediately, execution halts, leaving the initial value 1 as result. Since the initial value in registration is necessary for the multiplicative process, the mathematicians' convention that 0! = 1 seems deeply motivated.

The problem of semantic interpretation is another standard example of recursion, and not tail recursion. The information structure that it uses is the syntactic structure of a text, e.g. of a sentence; take this to be a dependency tree, with lexical labels at the nodes. For example, the dependency tree for "John ate a good breakfast" is

```
( ate ( John, breakfast ( a, good ) ) )
```

Such a tree can have any depth, and any number of dependents at a node. The meanings of all the dependents must be calculated before the meaning of the governor can be determined, just as the values of arguments must be known prior to execution of a function. The list of partial tasks can grow without limit.

Semantic interpretation can be performed by a binding of G, with a new kind of filler in the registration slot:

```
SemInt = G ( Compose Dict D-tree Dependents )
```

where D-tree is the dependency tree
Dict is a dictionary-lookup function, replacing the lexical label at any node with a semantic characterization
Compose inserts a list of meanings of dependents into the meaning of a governor
Dependents both keeps track of partial results and makes recursive calls

To illustrate, let the D-tree be (ate (John, lunch)). It is an actual list, and indexing gives the value "ate". Dictionary lookup finds the semantic characterization of this lexical item. Next, the registration slot becomes active at the control level. If the meanings of "John" and "breakfast" were registered, then Compose could insert them into the meaning of "ate"; but those meanings are not registered. The control type R2' effects the recursion by taking the name of a partially bound form--SemInt--from R0, completing the binding with the next element of the information structure, and invoking. This is a recursive call; the information structure passed is (John, lunch). If "lunch" had a dependent, another recursive call would follow, but the recursion halts on null information. At the end of the recursion, the meanings of the dependents of "ate" are in registration, Compose can do its work, and the interpretation is complete.

This illustration reveals the method by which iterative and recursive computations are distinguished: If the content of R is real, the computation is iterative; if not, then recursive. The illustration reveals more besides: Metagramming works by writing specifications for what does, can, or may exist; and by forcing enough computation to deliver results whenever possible. The specification can describe a mathematical function, an element in a database, or a program.

IMPLEMENTATION AND ELABORATION

A very small fragment of metagramming has been implemented. The method chosen resembled that of threaded code: In a compact segment of machine storage, pointers to definitions were stored. In fact, a threaded-code executive was used to run through these pointers. But the content of the storage area was modified during execution, as the foregoing discussion of G clearly requires. This method is close to standard practice, and can be used for further experiments and development.

The elaboration of the G form as a universal executive depends on ease of writing new indexing and control functions to fill 1-level and 2-level slots. Nothing prevents the use of bound G forms for this purpose; and with that technique, new kinds of structures can be considered for the information slot; for example, an interactive input language has a structure, and its interpreter can be inserted in I1'.

The G form, with four slots containing 0-level, 1-level, and 2-level fillers is itself an information structure; hence it is an object on which metagramming can work in a broad way. Because this form is invariant, or need exist only in a controlled diversity of types, operations on it should be more easily defined than operations on other kinds of program structures. These operations belong to domains of automatic and adaptive programming that have yet to be explored.

A PERSPECTIVE ON THE ART OF COMPUTATION

Metagramming is the art of creating information systems in which certain forms of expression become meaningful. This is in contrast with programming, which is the art of restating problems in a fixed form of expression. For practical application, the significant forms of expression are the languages of operatives, supervisors, and managers in business; of technicians, engineers, and scientists; and of officers of government, including military tacticians and strategists. These concise languages are appropriate for interaction with the computer; a machine that accepts only a basic, generalized language seems stupid to the user and therefore irritates, even if its thoroughness, speed, and other peculiar powers make it indispensable.

The technical languages of specialists take for granted different worlds, each with its own things, events, and logic; since what is taken for granted can be left unsaid, these languages provide exceedingly compact forms of expression. The information system that accepts a concise language as a programming language must contain a world model, including the logic that verifies internal consistency.

Certain world models are relatively simple. Basic accountancy deals with journals and ledgers; its principles of organization are the calendar and the chart of accounts, and its logic is the additivity of money. Present-value calculations, depreciation, etc., introduce multiplicative logic. Another simple world model is used in graphic arts--the simplest version sufficing for design of reports and forms. This is a world of sheets of paper, lines and columns of type, character heights and widths, and so on. But this world also contains text, with an independent structure of chapters, paragraphs, and sentences--or invoices with addressees, lists of entries, totals, etc. The logic of the graphic arts is the mapping of text into space on the page.

Other world models are, of course, enormously complex. Modern science is attempting the development of a coherent model encompassing every kind of observation. In the long run, metagramming could become a tool for science. It is not necessary, for the present, to consider highly complex world models. Many applications with simple structure need attention.

The world model of the specialist in computation has a unique role in metagramming. This is a world of internal and external storage, of machine operations, of input and output devices; and also a world of programs, files, and other information structures and processes. For metagramming to create an information system, it must use this model. A language of interaction with computers is meaningful only if it has an effective interpretation in the computational world model.

The creation of information systems giving meaning to new forms of expression is not new. It is, in fact, a phenomenon that has occurred many times in the history of human culture, on both large and small scales. And

the phenomenon occurs in the education of every child. The possibility of such creation is implicit in human cognition and language; the term 'metalingual', which applies to language about language, takes for granted just such creativity. In computing, work in artificial intelligence has produced a few world models, using methods invented on the spot.

The following paragraphs describe a structure of worlds that could be built. If this structure is created with the necessary skill and knowledge of the users who must occupy it, the software problem can be eliminated.

The initial system--Metagram 0--is simple; the method of threaded code demonstrates that there is no need to construct a rich, powerful precursor if the system itself can grow. The information structure of this system characterizes a primitive language. The processes realized by binding its functional forms construct and register the next system. (From the beginning, Carr and his associates thought of the growing machine as a tool for making new languages.) Like compiled code, the contents of registration is ready to execute.

Metagram 1 can give its operator good mnemonics for primitive operations, a decent syntax, methods of testing all or part of a new program, and reasonably good display facilities. Now, it is inherent in metagramming that the Metagram 1 operator can be isolated absolutely from Metagram 0; but such isolation is not obligatory. The system-wide capacities of Metagram 0 can be replicated in a Metagram 1 system, with the convenience that this level gives. The operator at this level is the designer of one or more Metagram 2 systems, each realized by building it in a registration slot.

From Metagram 0 through Metagram 6, the worlds constructed become more specialized, more diverse, with data structures and procedures that take on the particular characteristics of, say, an industry and then a single company, or of a technical specialty and then of a single technical department. Each system is composed of just what is needed, rather in the way an operating system is sometimes generated to suit a unique installation. Hence the typical system can be relatively simple, small, and fast.

One might expect a very large number of different systems at level 6, and therefore feel that metagramming loses the value of standardization. Two factors mitigate the loss. On the one hand, the main organizing principles of the system can and should become standard; the notion of a universal executive is in this direction, and many elements and features should be used widely. On the other hand, diversification of systems should correspond well with lines of responsibility. If the goal of standardization is that certain competent persons should be able to read and modify any code in use anywhere, the goal is unrealistic; to understand code, one must know the application as well as the system, and no one can know every field. From the perspective of metagramming, what is wanted is the application specialist for whom the right computing system is transparent.

And this plan shares with threaded code the advantage that only a small part of the structure need be modified by experts to suit different machines; from that kernel, the adaptation of the remainder is automatic.

INFORMATION AND ANALYSIS

The intelligence requirement of a major government is comparable in scope to the intellectual purposes of mankind: To know and understand the world in which we live. The priorities of a government are often different from those of science or scholarship, and intelligence customarily delivers an appraisal of a case whereas science customarily delivers laws or universal theories; but these are not absolute differences. Making full allowance for the differences, one can still apply the history, sociology, and philosophy of science to intelligence and begin to see that this enormous and eclectic field can have a disciplinary structure.

Three issues stand out in the open literature.

1. Volume of information.
2. Incompatibility among sectors.
3. Interdependence of presuppositions and conclusions.

These issues arise both in general intellectual work and in governmental intelligence. From the perspective of metagramming, they are not problems to be solved so as to make intelligence analysis (or scientific research) easy, but terms of reference for the organization of work that is difficult and will remain so permanently. If these are not problems that will be solved, then intelligence analysis (like general intellectual work) needs a perspective broad enough to encompass them. The new perspective is one that can look at and deal with abstractions.

VOLUMINOUS INFORMATION

"The extent of the informational processes of the Department of Defense is staggering," wrote Frederick B. Thompson early in the 1960s (ref below). He made numerical estimates, claiming only that the "logarithms are of the right order of magnitude": 100 billion billion "Considerations within area of responsibility" for DoD, with "Average time between significant contextual changes" 1 microsecond. Surely after twenty years these figures have grown even more staggering. The Government of the United States has still more to consider. The computing capacity of the Government is an indicator of the volume of information over which control is attempted.

Thompson depicted the command hierarchy as facilitating control over this great quantity of information. Those at higher ranks have "to leave certain of these details to subordinates and to deal with higher-level abstractions..." Conversely, those at lower ranks accept definitions of the context

Ref: Thompson, Frederick B. Design Fundamentals of Information Systems. In MILITARY INFORMATION SYSTEMS: The Design of Computer-Aided Systems for Command, edited by Edward Bennett et al. 1964. Pp. 46-87.

of their operations as fixed at higher levels and issued as commands. Thompson's characterization of abstraction is "grouping otherwise discriminable aspects as a single object." This is generalization; true abstraction might instead be phrased as "organizing otherwise unrelated aspects into the pattern of a single object or event".

Granting Thompson accuracy even for the order of magnitude of the logarithms of his estimates, it is clear that true abstraction is part of the informational function of hierarchies of command. Where 10 to the 12th or 14th power details are involved, only abstraction--not generalization--serves. In the contemporary military establishment, two or three levels of abstraction (somewhat corresponding to the break widely recognized at the Colonel's rank, and to the distinction between strategy and tactics) are in use. Abstraction moves the thinker to a new world; see the discussion of science, pp 10-13. The most senior officers with broadest responsibilities live in a different world from the most junior officers with strictly local areas of responsibility; and a third, intermediate world may exist. The personnel of different grades appear to repeat, in contemporary life, the modes of thought of stages of cultural evolution: Lore is sufficient for the operative with concrete tasks, engineering is a suitable mode for the tactical officer, strategic planning requires cognitive skills of the scientific level, and at the highest levels in the Department of Defense the necessary abstractions are so high that only the most powerful systems of thought available can suffice.

In cultural evolution, increasing depth of hierarchical organization is a strong correlate of the overall level of the society; it is one of Raoul Naroll's (ref below) indicators of level, along with settlement size and occupational specialties. Assuming that the major stages of cultural evolution are marked by increase in the number of levels of abstraction available in the local cognitive system, Thompson's argument shows how the command hierarchy brings all of the available levels to bear on the informational problems of complex action. Hierarchy has in addition the social advantage of permitting each commander to know as individuals all immediate subordinates, and the psychological advantage of making each person directly responsible to a single superordinate. These advantages are perhaps more generally recognized, but the cognitive value of sorting out levels of abstraction is immediately obvious.

The informational function of the hierarchy is not summarization but translation. The field reports have to be recast in strategic terms to be of use to the highest levels of command; totals and averages are not enough. The problem of data volume is not accidental but essential in military and political structure. Data reduction in the mode of generalization cannot serve; the mode of abstraction must be examined.

Ref: Naroll, Raoul. What Have We Learned from Cross-Cultural Surveys? AMERICAN ANTHROPOLOGIST 72:1227-1288, 1970.

INTERSECTOR INCOMPATIBILITY

Governmental and, in particular, military organizations are large. In the hierarchy, some subordination of responsibility is merely territorial: A large command is divided geographically. But many subordinations are by technical specialization, as, for example, the Strategic Air Command is given the technical problem of action at a great distance from its bases. Where each unit has a computer but interconnection is required, the technical specialties raise problems of compatibility.

No doubt some of the problems of compatibility between computer systems arise on account of competition among makers of computers and software or among users. And no doubt other compatibility problems have arisen because systems were installed before the significance or even the possibility of interlinkage was recognized; independent designers made independent inventions to roughly the same ends, leading to differences in crucial detail.

Nevertheless, incompatibility is to be expected as a permanent trait of a complex with technical specialization. It is proper and necessary that each specialized command of whatever rank represent in its own computing system the world in which it operates. This world consists of categories of things and events which are not identical with, and if the technical specialization is sufficiently sharp and deep not even comparable with, the categories of things and events in some other world--even if the two worlds occupy the same region in time and space. The categories of each command are necessarily reflected in the rubrics of its data and in the processes of its software.

What specialized commands do have in common is their superordinate. The higher headquarters must deal with all of them. Its terms are more abstract, but part of their responsibility is to translate between its terms and their own. Thus the only natural way to pass information laterally between subordinates is via the common superordinate. The volume of data or the need for timeliness may make this route impracticable; but reference to the more abstract language of higher headquarters is the natural regulator of lateral communication even so.

The problem of abstraction thus arises when information passes between two commands on the same level when they implement (make more concrete) the abstract requirements of their common superordinate in technically, theoretically different ways.

PRESUPPOSITIONS AND CONCLUSIONS

One of the standing problems in the philosophy of science is the impossibility of establishing any fact, at any level of abstraction, without making presuppositions of many kinds: Taking much for granted, the scientist discovers a fact; but taking something else for granted, the scientist could have discovered quite a different fact. The analyst of intelligence, like the commander in the field, is subject to the same problem. The commander must accept the context defined by higher headquarters with limited exceptions; but intelligence, like science, is responsible for its own presuppositions.

For the scientist or analyst working in what Thomas Kuhn called the "normal" mode (ref below), the problem is discounted. The presuppositions constitute a "paradigm". From textbooks and through apprenticeship, the trainee learns the commonly accepted facts and methods of work. The mature worker contributes to knowledge by applying the accepted methods and arriving at a new fact of the kind in the books; the intelligence analyst might determine the location of one base or weapon in a region already known to contain others like it.

But there is another mode, which Kuhn called "revolutionary". An example from science is the shift from geocentric to heliocentric astronomy. An example from intelligence work is the discovery of Russian missile sites in Cuba. In scientific revolutions, presuppositions change. Since many of the presuppositions of science concern nature, whereas those of intelligence concern other governments, revolutions are probably more common in the latter. The issue is, under what circumstances do scientists (or intelligence officers) alter their presuppositions? And are those circumstances appropriate? Graham Allison (ref below), relying on a report of the House Intelligence Committee, writes that "understanding of Iran's internal situation" was made difficult by assumption of the Shah's stability. Whether we can trust science (or intelligence) depends in large part on the interplay between presupposition and conclusion.

Examining the problem of scientific knowledge in the 1930s, Karl Popper (ref below) concluded that we have no means of arriving at absolute truth, but that we are more justified in holding to facts (or methods) insofar as we have tested them. That is, testing eliminates error, even if no humanly feasible series of tests can establish truth. Different readers of the history of science do not agree on the significance of failed tests in scientific revolutions. As philosophy, i.e. as a guide to effective conduct, Popper is not much challenged. But the fact may well be that sociological, aesthetic, and other factors influence the outcomes of attempts to revolutionize science.

A partial survey of the literature, including Toulmin and Campbell (refs below) as well as Popper and Kuhn, does not bring to light much on abstraction. The paradigm concept suggests an abstraction, and other hints appear. It seems worth while to say explicitly that the scientist (or analyst) must know and use several levels of abstraction jointly. The most abstract level can be called the thinker's own system of thought, and access to it is

Ref: Kuhn, Thomas S. The Structure of Scientific Revolutions. University of Chicago Press, 1962.
Allison, Graham. An Intelligence Agenda. New York Times, 21 Dec 1980, page E-17.
Popper, Karl. Logik der Forschung. Julius Springer, Vienna, 1935. Translated as The Logic of Scientific Discovery. Basic Books, New York, 1959.
Campbell, Donald T. Evolutionary Epistemology. In THE PHILOSOPHY OF KARL POPPER, edited by P. A. Schilpp. Open Court Publishing Co., LaSalle, Illinois, 1974. Part I, pp. 413-463.
Toulmin, Stephen E. Foresight and Understanding. Harper and Row, New York, 1963.

difficult or, more probably, impossible for the thinker. The most concrete level is the level of sense data. As Campbell points out, neither of these levels is easy to alter. If these two levels constitute the entire cognitive system, the thinker is in a primitive mode corresponding to the lowest level of human culture. A third level, between the most concrete and the most abstract, puts the thinker at the level of Aristotle: The new level is a reconstruction of the former system of thought, but appears to the thinker as an analysis of nature. A system with four levels might be described as having a sense-data level, an objective level, a subjective level, and a system-of-thought level; this is the level of the Renaissance and the science that developed between roughly 1500 and 1900. In the present century a fifth level seems clearly to have developed; perhaps such a term as "relational" or "interactional" is appropriate to it.

For the typical scientist described by Popper, Kuhn, Campbell, or Toulmin (among others), the sense data were clearly distinct from the abstract concepts of objective theory, and the objective theory was clearly distinct from the subjective observer and theorizer, who could make mistakes in seeing and recording the data. The separation between objective and subjective levels was probably necessary to the formulation of a heliocentric theory of the universe; as long as objective and subjective models remained confounded, the idea of a moving earth and a (relatively) stationary sun could not be formulated. A certain relationship between sense data and objective model was recognized: Salva apparitionis, 'saving the appearances', was the phrase, and it meant that the model must account for the data. But the relationship between the subjective and objective levels was not understood in any depth; indeed, it is the aim of Popper and others in this century to understand that very relationship and so to fill in the new relational or interactional level of abstraction.

This model of cognitive evolution suggests a new view of revolutions in science. The instigator is a person who has begun to think with one level of abstraction beyond those of contemporaries, has begun in fact to operate on a reconstruction of what for others is the system of thought itself. No system of thought has ever been perfect, and small operations on any system of thought can surely make apparent difficulties and imperfections. Thus anyone who could begin to separate subjective and objective models would have been in a position to express doubts and concerns that others had felt without having the means of expression. Such a person would also have new capacity to manipulate the objective model--to consider the earth moving around the sun, for example. And the combination would be effective in a way that both Popper and Kuhn miss.

The significant point is that every level of abstraction is bound on both sides. The dictum of 'salva apparitionis' concerns but one side: The objective model is bound to the data. But every model is also bound to the more abstract side as well, for the sake of logical integrity if nothing else. Before the Renaissance, the single intermediate level of abstraction would have been bound to sense data and to the system of thought of the European thinker. Upon the emergence of another level, the model of the universe is still bound to the sense data on one side, but to an explicit system of observation and calculation on the other--and it is this latter that is bound to

the system of thought. The elegance of the universe--geocentric, with orbits composed of perfect circles--gives way to elegance of methods in observing and calculating. And elegance can be understood as pleasing to the system of thought, both before and after the revolution.

The twentieth century has a new level of thought. Methods of observation and calculation are no longer directly answerable to unconscious standards; elegance is now the appropriate criterion for concepts in the philosophy of science, which imposes its explicit tests on method. But these new tests are only beginning to emerge, and their nature is as yet uncertain. To anticipate their nature, a final evolutionary sequence seems helpful: What is the conservation principle that regulates thought at each stage? The user of lore conserves effectiveness; the growth of lore is pragmatic. Aristotle's principle seems to be the conservation of category, in the sense that each kind of entity is capable of certain kinds of action. The next stage clarifies the subject-object distinction and begins experimentation; the principle of conservation of susceptibility, that each natural kind responds to operations in its own manner, seems to guide scientific thought. The present stage deals with systems of logic as well as with systems of observation; it makes recursive analyses of nature, over indefinitely many orders of magnitude of time, space, and numerosity of elements. A principle of consistency through levels seems to be what we use: Conservation of structure-function relationships.

Allison (ref p. 43) asks "How can the intelligence community's analytic competence be substantially enhanced?" His answer is to "deepen the expertise" of analysts and to enlarge the community. A more significant answer is to solidify and apply the systems of thought that are growing up in contemporary critiques of science. A new perspective on computer software, such as that implied in metagramming, and a new perspective on intelligence analysis could combine to produce systems at a new level of competence.

Appendix

COMPUTATION AND STRATEGIC INTELLIGENCE

Notes on Sherman Kent, Double Contingency, Gibsonian Psychology, and Metagramming

William L. Benzon

SUMMARY

The theoretical expansiveness of metagramming paradoxically serves the purpose of reducing the problem of writing software to one which can reliably be solved.

The goal of the intelligence analyst is to detect and analyze substantive problems against a background of descriptive, reportorial, and speculative-evaluative knowledge; these forms of knowledge are stratified, with higher strata being meta to lower strata.

The analyst needs a metagramming system with the capacity to check the prima facie plausibility of causal paths involving double contingency calculation; this checking conserves the integrity of structure-function correspondence in a recursive analysis of causal paths.

Gibsonian psychology frees us from Cartesian solipcism and doubt and provides a criterion for ascertaining the reality of our perceptions: If new information becomes available when the object is inspected, then it is real; otherwise it is a figment of someone's imagination. Invariance detection is the medium of perceptual interchange with the environment and conservation principles are invariance principles.

Just as the search space in which chess is played is meta to the one generated by the rules of chess, so the space in which computers can reliably be metagrammed is meta to the one in which computers have heretofore been designed and built. The higher level space is related to the lower through representation functions which work well with complex irregular objects.

Metagramming works by inducing an ecological closure over an otherwise unbounded search space. An account of the application domain (e.g. strategic intelligence) is crucial to formulating the closure.

INTRODUCTION

The purpose of the present section is to juxtapose a few relatively informal discussions of intelligence, psychology, and metagramming in a way which is more suggestive than conclusive. The selection of suggestiveness rather than conclusiveness as the rhetorical aim of this report is quite deliberate and grows from the peculiar difficulties involved in explaining just what metagramming is and what it has to offer the intelligence analyst. Con-

clusiveness in this domain demands technical detail; and technical detail is intelligible only if one understands what the detail is for and why it must be that way and not another. Without that understanding the technical detail degrades into a pile of twigs, leaves, and branches unrelated by the more inclusive concept of the forest. This report is meant to suggest the forest by means of a few conceptual strokes in much the same way that the Chinese landscape painter suggests worlds of mountains and forests with a few quick strokes of his brush.

The difficulty is that metagramming is a new, and therefore strange, beast. It is not an extension of programming techniques; but an alternative to current philosophies of programming, of organizing computers to meet human informatic needs. From the programmer's point of view the problem addressed under this contract, that of providing a common access language for databases in a network, is a complex but nonetheless limited one; one that certainly doesn't justify excursions into cultural evolution and perceptual psychology. From the metagrammer's point of view these matters must be considered if the problem is to be reduced to manageable proportions. That is, what the programmer sees as illegitimate and pie-in-the-sky theorizing is, for the metagrammer, a necessary prerequisite to specifying a solvable problem--which is rather different from a problem we can hack away on until something happens. The metagrammer sees the programmer's strictly focused practicality as leading to an unending stream of mutually incompatible local solutions to local problems which will never add up to a global solution, as an unordered pile of twigs and leaves which will never be a forest no matter how big the pile becomes. In contrast the rather freewheeling theoretical creativity that has been the major activity under this contract is the only route to a practical solution of the problem of accessing databases--and by 'practical' I merely mean a solution that works.

Thus when some hypothetical programmer accuses some hypothetical metagrammer of making a mountain out of a molehill, that metagrammer replies, "Yes, but I have done so, not by piling on more and more dirt, not by collecting many molehills in one place so that their mass approaches a mountain's, but by adopting the perspective of the ant so that I could gain intimate knowledge of the structure and functions of the molehill." The purpose of these notes is to suggest what comes to view when one adopts the ant's perspective.

In the first section following, I consider Sherman Kent's STRATEGIC INTELLIGENCE FOR AMERICAN WORLD POLICY (ref below) and conclude that the goal of the intelligence analyst is to identify and analyze substantive problems against a background of alternative scenarios. In the second section I consider an analogy between the detective's situation and the intelligence analyst's and conclude that the analyst must make double contingency calculations. Gibsonian perceptual psychology moves into prominence in the third section and leads to an ecological interpretation of metagramming. The fourth section views metagramming as a technique for transforming a search through an unbounded space into a search through a bounded space. And a concluding section rounds things off.

Ref: Kent, Sherman. Strategic Intelligence for American World Policy. Princeton University Press, 1966.

SHERMAN KENT ON STRATEGIC INTELLIGENCE

Sherman Kent's STRATEGIC INTELLIGENCE FOR AMERICAN WORLD POLICY was first published in 1949. While much has appeared in the open literature on intelligence which supplements Kent's discussion, nothing has been broad, deep, and subtle enough to replace it. It is thus a suitable point of departure for these notes, where I am most interested in his discussion of intelligence as a form of knowledge and the methodological problem of identifying and analyzing the substantive problem.

To begin, it is obvious that anything which is humanly knowable is of potential value to the intelligence analyst. As a practical matter, most of that information will not be needed, but there is no a priori way of determining what is essential and what is not. Thus the job of organizing information for the analyst is basically one of imposing a partial ordering on all of human knowledge such that the knowledge most relevant is at the top of the ordering. In fact, we need a nested set of partial orderings. The most global ordering determines what information will be held in computer databanks and what will be excluded. The most local ordering is that created when the individual analyst asks a specific question. Intermediate orderings reflect intermediate levels of demand, e.g. the typical needs of a group of analysts over a six-month period.

In Kent's scheme this knowledge is organized at three levels, descriptive, reportorial, speculative-evaluative. The descriptive level is simply an encyclopedia of basic information. The reportorial function keeps track of current events; it updates the encyclopedia. At the speculative- evaluative level the governing question is, to quote Kent (p. 40):

> What knowledge should the U.S. have about the future of other states in order to have the requisite foresight?

Clearly one must use sophisticated inferential skills at the speculative-evaluative level.

From the metagrammer's point of view these three levels are meta to one another. Concepts at the speculative-evaluative level are about the relationship between information in the encyclopedia (descriptive) and current reportorial information. These relationships are used, on the one hand, as the basis for inferences about the future. The encyclopedia provides the background against which the current state of affairs is interpreted and projected into the future. The modes of interpretation and projection are internal to the speculative-evaluative function.

On the other hand, speculative-evaluative concepts are used to set specifications for current reporting and encyclopedia building. For no encyclopedia can ever be complete and no team of reporters can keep track of all current events. Some things will be in the encyclopedia, but much will be missing. Some streams of current events will be monitored, but most will be left unwatched. The requisite selectivity will be determined according to the informatic needs at the speculative-evaluative level (which include, by the

way, a need to be open to unexpected events) and then realized in processes at the reportorial and descriptive levels.

Moving the analysis down a step we see that proper performance of the reportorial function requires concepts about the descriptive level. For the encyclopedia is organized according to some scheme, which consists of concepts about the information in the encyclopedia. To add information to the encyclopedia one must know the scheme by which it is organized; hence the conceptual scheme of the reportorial function must be meta to the languages in which the encyclopedia is written.

A metagrammed realization of this organization would grow from the bottom up. But all levels must be present from the beginning. And there is no reason to expect that processes at the lowest level would ever become so routine that no human intervention at that level would be necessary. Rather there is a gradual shift in which the greatest concentration of man-machine interaction shifts from the descriptive to the reportorial and then to the speculative-evaluative level.

The other aspect of Kent's account which is most crucial to our understanding the intellectual structure of strategic intelligence is his formulation of the analytic process. It has seven stages:

1. A substantive problem is detected.
2. A problem description is created.
3. Data are collected according to specifications set in 2.
4. Data are evaluated.
5. Data are analyzed to discover their latent meaning.
6. More data are collected according to specifications set by hypotheses generated in 5.
7. When all hypotheses but one are probably falsified, the one remaining is presented to the consumer.

Notice that we have two data collection steps, 3 and 6. This type of looping back suggests that we're dealing with a process best analyzed in terms of a metalanguage operating on an object language where the metalanguage is either conducting a recursive search through the object domain or is setting servomechanical reference levels for the object domain comparators (as in William T. Powers, BEHAVIOR: THE CONTROL OF PERCEPTION; ref page 11).

The recursive search interpretation seems germane to designing a metagramming system for the analyst in which the system could search the data base in fairly routine ways to establish the relative credibility of certain routine and well understood explanatory patterns. Such a system would, of necessity, be a relatively mature system, for it would take time to build up a useful library of explanatory patterns.

The servomechanical interpretation seems more applicable to the activities of individual analysts and to the whole intelligence community. Data are being gathered at all times; hypotheses are being generated at all times; and finished analyses are being presented to consumers at all times. The in-

dividual analyst must consider the data he has at hand; relate them to the hypotheses he can generate; and consider the needs of his client--and all are ongoing activities. The consumer's purposes regulate the range of hypotheses which are most salient; and the range of hypotheses regulates the retrieval of information from the encyclopedia and the monitoring of current affairs. Descriptive, reportorial, and speculative activity are going on at all times and in close mutual regulation; a linear characterization of such a process, such as Kent's seven steps, fails to capture its essence.

This brings us to the critical problem for the analyst, the detection and analysis of a substantive problem. Of all the pieces of information which pass before him, of all the pieces of information which could pass before him if he needed them, which of them specify an unfolding pattern of events which will have a significant effect on national security? Certain sorts of things are obviously important--large troop movements, massive crop failures, the discovery of large deposits of scarce minerals--but other patterns don't have such obvious signs. How do you detect them?

Sherman Kent gives the following answer (p. 160):

> The only answer lies in picking a man who already knows a good deal about the substantive area in which he is supposed to ask questions, and who has an inquiring mind; and then see to it that he has ready access to every scrap of new incoming evidence on it, access to everyone who knows about it, and freedom from other burdensome duties. But if you go below the surface and ask, how does one come to ask oneself good questions, you start down one of the main roadways of epistemology. It is not my intention to do so.

The answer is a wise one. But it is not very comforting in a world where genuinely wise men and women are few and far between.

But if we understood something about the nature of the problem these wise men face, perhaps we could come to understand enough about their methods of solving such problems to be able to give them a computer aid which would extend their analytic range. A similar aid would also extend the range of their less gifted and experienced colleagues.

I think we can go a step beyond Sherman Kent by placing the detection problem in a more tractable form:

> THE DETECTION PROBLEM: The goal of the intelligence analyst is to detect and analyze substantive problems against a background of alternative scenarios.

The trick is to realize that while alternative scenarios are in part constructed from the results of the analyst's work--they are produced at the speculative-evaluative level--they are also part of the background he needs to do his work. Scenarios are background because they tell the analyst what to look for, i.e. those events which will tell just which scenario is actually being realized. Without this guidance the analyst faces a mass of information

without any way of determining what is worthy of his attention. Yet formulating alternative scenarios requires the results of the analyst's work--one wants, insofar as possible, to base scenarios on the current affairs rather than on one's imaginings about those affairs.

The next section concerns conservation principles which guide the formulation of hypotheses and scenarios, while the section after that considers the problem of distinguishing between the real and the imaginary in the context of Gibsonian psychology.

INTELLIGENCE AS DETECTION

Let us begin by considering the way in which the intelligence analyst's job resembles and differs from the detective's job. Both must sift through an indefinitely large set of objects and events to find just those sets which yield coherent causal patterns. However, the detective generally doesn't begin his work until a case presents itself to him; his job is to solve the case. But the intelligence analyst must detect emerging patterns early enough so that the operational branches of the government can prevent them from becoming 'cases'. Further, the intelligence analyst must routinely make double contingency calculations while the detective has to do so only rarely. This latter difference will turn out to be deeper than the former.

We can start with the detective and his case. A case is any irregularity, any rupture, in the fabric of social existence. A child disappears, a cigar store is robbed, the corporation's books don't balance out, these are all obvious disruptions in the fabric of social interchange. In each case the detective's job is to find the causes and use that knowledge to repair the torn fabric.

To do his job the detective searches for clues and elaborates a theory of the case. But objects and events don't obviously proclaim themselves as clues. The most obvious clue is the crime itself. Beyond this, whether something counts as a clue or not often depends on the current theory of the case. If one thinks the child has run away, then neighbors' reports of yelling and screaming between parents and child count as clues to the child's motivation. If one suspects a sex crime, then reports of a stranger in the neighborhood are more important. Objects and events thus count as clues to the extent that they fit some current theory of the case. But any theory of the case depends on the clues which suggest it. We thus have the same suggestive circularity we found in Kent's seven-step analytic sequence. Clues exist at the level of data while theories of the case exist at the level of analysis and hypothesis formation.

In actual practice certain things are obviously clues--this is known from past experience. And these clues, plus general knowledge of crimes and criminals, can be used to suggest several theories. Each theory in turn suggests its own set of clues. The ensemble of current theories collectively imposes a partial ordering on objects and events which have clue value. The detective proceeds by searching for clues in order of clue value and revising his ensemble of hypotheses (and thus his ordering of events and objects

according to clue value) by what he finds. The process of solving the case thus proceeds in a space bounded by the detective's repertoire of theories and the objects and events those theories specify as clues.

Now let's consider the analyst. Perhaps more often than not, he does not have a specific case, an obvious rupture of the social fabric, before him. And even when he is offered that luxury, he can't consider the rupture in question (e.g. the assassination of a diplomat) to be the end point of the causal pattern he must reconstruct. Rather, that rupture is only a clue to some pattern whose purpose can only be guessed at and whose terminal point could well be months or years in the future.

This difference, however, is purely quantitative--the analyst faces larger causal patterns and so may have to consider a larger ensemble of hypotheses each of which ranges over a larger number of clues. There is also a qualitative difference. To figure out what significance a piece of information has for him, not only must the analyst figure out what significance it has for other nations, but he must do so with a process which allows for double contingency: The significance it has for them depends on the significance they think it has for him, which in turn depends on the significance he thinks it has for them, etc.

This is double contingency calculation, discussed by Talcott Parsons in THE SOCIAL SYSTEM (ref below). The threatening regress is obvious enough. Such calculations are difficult, exhausting, and never conclusive. Hence much of social life is lived according to roles which specify routine interaction patterns, thus eliminating double contingency calculations. But the routine depends on trust; where trust is absent, double contingency calculations are as inevitable as bluffing in poker (which involves double contingency). It follows from this that the patterns the analyst is looking for are those created by double contingency calculations, for those are the most threatening patterns, the ones most likely to result in unconstrained violence. The detective deals in known ruptures of the social fabric; the analyst must detect those patterns which mean the absence of any social fabric, the absence of trust.

This is a more subtle point than that embodied in the obvious enough principle that one pays more attention to the activities of groups which can't be trusted than to the activities of trustworthy groups. For the attribution of trust and distrust is not external to the social process; it is part of the process. Just as the attribution of clue value to objects and events depend on the theory of the case, so the attribution of trust value depends on that very pattern of interchange which will be modified by that attribution. And the analyst is part of that interchange. If he reads his evidence with a distrusting eye he can see threat everywhere. And the trusting eye can as easily see a social fabric where there is, in fact, only naked distrust.

Part of the solution of this problem is to be found in conservation principles, with which this section concludes. Another part requires a foun-

Ref: Parsons, Talcott. The Social System. The Free Press, Glencoe, IL. 1951.

dation in Gibsonian psychology--which will be taken up in the next section of this report.

Conservation rules (see page 45) constrain the creation of patterns. For neither the detective nor the analyst suffers from the lack of patterns applicable to the data available. On the contrary, finding patterns is easy. But finding plausible patterns is difficult. (The difference between these two has found its way into popular culture as the Rube Goldberg device.) Conservation rules constrain the search for patterns to those which may fruitfully be submitted to Gibsonian reality testing (see next section). The conservation principle appropriate to the detective's task is Karl Popper's falsification criterion. The intelligence analyst needs the more powerful principle of the "conservation of the function-structure relationship in a recursive analysis of natural phenomena".

Popperian falsification enters Kent's account of the analyst's methodology at the seventh step. The logic is familiar. Generate alternative and mutually inconsistent hypotheses to explain some phenomenon. Then generate a series of tests which will discriminate among the hypotheses. If the outcome predicted by a particular hypothesis fails to match the experimentally determined outcome, then that hypothesis is falsified. The falsity of predicted outcome is inherited by the hypothesis which generated the prediction; falsity is thus conserved.

This seems adequate for most of the detective's needs, but it will not do for the intelligence analyst. For the evidentiary value of much of his data depends on whether or not it was intended to deceive him. If the data are veridical then they can be used to falsify hypotheses. But if they are part of a deception, then they loose much of their value. Determining whether or not one is being deceived is often difficult. And believing that one is the object of deceptions and hoaxes has the seductive attraction of bringing order and coherence into otherwise incoherent patterns of events. Hence belief in malign conspiracies is easily engendered and dislodged only with difficulty. What conservation principle can help the analyst to protect his perceptions against illusory deceptions and conspiracies?

As indicated above, I believe that conservation of the function-structure relationship is needed. But the principle is new and its explication in the context of such a difficult example is difficult. Hence what I have to say on this matter is likely to be unfortunately and unintentionally vague.

Let us begin with a more accessible example, that of neural models. Here we want to insure that computational functions are distributed among systems and subsystems in the same way in the nervous system and in the model. The functional forms of the model must correspond to the functional architecture of the nervous system in a straightforward way. This correspondence can be assessed at every level from the most macroscopic characterization of functional regions down to the operation of the individual neuron. The correspondence is subject to Popperian falsification at each level; but there must also be correspondence across levels. It is this cross level correspondence which is conserved recursively.

To get back to the analyst's problem, let's consider a specific example which is a bit closer to the sort of analytic problem he faces. It has recently been argued, by a man in his book, the names of which escape me, that the moon landings are a hoax; they never really happened. It is easy enough to give this idea superficial plausibility--after all, few of us have any direct evidence on the matter and the indirect evidence we have, written accounts, photographs, motion pictures, etc., can be faked. But we feel intuitively that the extension of these sorts of causal chains would soon become hopelessly and implausibly extended and convoluted. That intuition is conserving the function-structure relationship through intersecting intentional and causal paths. A systematic and consistent doubting of the moon landings would require a doubt that extends so far beyond the space program that there would be no reliable evidence on which to base any conclusion on anything. The world might indeed be in such bad shape; but if it is, then it seems unlikely that any course of action can be argued for. Otherwise it seems best to rule such hypotheses out of court simply because they can't be investigated in any coherent way. The intelligence analyst needs help in determining which doubts would undermine the coherence of most of what he knows and which ones form a coherent pattern.

A mature metagramming system could help the analyst by applying the appropriate conservation principle in routine cases. The analyst might indicate a pattern which he believes to be plausible and the computer could check the consistency of this pattern with other patterns currently held to be valid. The results of this check could then be used to order the information in the system according to its relevance to the analyst's current problem.

In the absence of a mature metagramming system a careful formulation of conservation principles can help the analyst to notice those patterns which should be incorporated into the growing stock of functions available to the metagramming system. The idea is to so order the analyst's knowledge that that which is routine can be moved into the machine with minimal difficulty. In an immature system much of the conservation checking will be done by the analyst. But as the system matures it has its library of analyst-regulated conservation checks to serve as examples and it can, under the analyst's guidance, extend those examples to routine situations and so increase its library still further. After a while it may be ready to internalize principles for extending accepted examples to new cases. At this point it will be doing routine searches on its own. When a search turns out not to be routine, to be irregular, the machine will call on the analyst for help. And perhaps this irregularity will turn out to be an important one, a substantive problem.

Thus some of the analyst's wisdom may have become imparted to the machine. Or at least the machine has learned to organize information so that it can present the analyst with an environment in which he can readily apply his wisdom. But we need more. Conservation principles restrict search, but they aren't foolproof. As Heinrich Zollinger (ref below) has recently characterized at least one point of difference between Karl Popper and Thomas Kuhn, at least some scientific theories are abandoned without having been falsified.

Ref: Zollinger, Heinrich. Logic or Psychology of Scientific Discovery? CHEMISTRY IN BRITAIN 16:257-259, 1980.

A reading of J. J. Gibson suggests that those theories might have been given up because they failed to reveal anything new. Thus a plausible theory must not only be conservative, but it must also generate new perceptions.

THE RELEVANCE OF GIBSONIAN PSYCHOLOGY

Beyond its merit as an account of visual perception, the line of reasoning followed by J.J. Gibson in THE ECOLOGICAL APPROACH TO VISUAL PERCEPTION (ref below) suggests an Alexandrian approach to the Gordian knot of Cartesian solipcism and skepticism which has been with Western thought ever since Descartes wrote his MEDITATIONS. The Cartesian subject is trapped in a world limited to his own thoughts and perceptions, an asocial world in which his is the only mind, a world without validity tests. Gibsonian psychology gives us a subject who has direct perception of the world, who can share that world with others by walking the same paths they do (as they walk the paths he walks), and who can tell whether or not he is dreaming or hallucinating by moving his eyes and by manipulating the objects of perception. Artificial intelligence is a Cartesian discipline; metagramming is Gibsonian. The difference is of obvious significance to the intelligence analyst.

The Gibsonian perceiver inhabits an ecological niche whose structure is a function of the correspondence between the perceiver's nervous system and the external environment. That correspondence arises through the familiar evolutionary mechanisms. Detection of invariance is the medium of perceptual exchange between organism and environment. Those invariances are functions of the organism and its niche.

The basic perceptual act is called specification; in the psychological theory which underlies metagramming we talk of physiognomic perception. And where Gibson talks of representation we talk of propositional reconstruction of physiognomies. This notion of reconstruction is almost absent in Gibson, but it resembles the process of reflective abstraction which Jean Piaget has elaborated in, e.g., BIOLOGY AND KNOWLEDGE (1971).

Humans, and the natural languages they speak and write, perform physiognomic operations well; this is a matter of semantics, of selection. It requires a logic of similarity and difference, of opposition and contrast. Computers, and the languages developed for programming them, perform propositional operations well and easily; this is a matter of syntax, of grammar. And the appropriate logics have been created in the wake of investigations by Boole and Frege, Russell and Whitehead, Tarski and Church. A metagramming system must deal with propositions and physiognomies.

The best way to understand this distinction is to consider some examples. Looking through a pile of photographs and recognizing the objects, persons, and events in those photographs is a physiognomic operation. And how many hundreds of thousands of photographs could you recognize? In contrast, providing a verbal description of any of the photographs, either with the

Ref: Gibson, J. J. The Ecological Approach to Visual Perception. Houghton Mifflin, Boston, 1979.

photograph in front of you or from memory, is a propositional task. Consciously controlling your body while learning some complex muscular task, such as playing a musical instrument, is a propositional operation; executing the same action without even thinking about it is physiognomic. Following a boxing match is physiognomic; giving a running commentary requires translation from physiognomic to propositional. Intuitions which guide in the solution of problems are physiognomic; explicitly formulated procedures are propositional. Computer code must have a propositional form; but figuring out which hunks of code are needed for a particular application involves both physiognomic and propositional modes of thought. Physiognomic operations tend to be holistic, one might even call them gestalt, and, if not exactly unconscious, then at least they are preconscious. Propositional operations tend to be atomistic, with discriminable parts and subparts, sequential, and require conscious effort.

Further insight can be had by considering the strange friend phenomenon. You see a friend but notice that something has changed. You examine the friend closely to find out just what changed. You may succeed in spotting the shaved mustache or the new haircut or you may fail utterly and have to be told about the change. Noticing the change in the first place is a matter of physiognomic perception. The attempt to specify the nature of the change takes place under propositional guidance. That we can often notice the change without being able to explicate it indicates the power and the limitation of physiognomic perception--it picks out change quickly but doesn't specify the change exactly.

Consider one last illustration, geometry. The straightedge and compass procedures for constructing simple geometrical objects are a propositional reconstruction of the visual physiognomies which specify those objects. The axioms and theorems of geometry are, in turn, propositional reconstructions of those drawing procedures.

Now we are ready to take up the topic of invariance, which leads naturally to the conservation principles of the previous section. When that has been done we can conclude this section with Gibson's reality criterion.

Objects and events engender different phenomena at the surface between organism and niche depending on the relationship between the organism and the objects and events it is perceiving. By taking appropriate account of the various streams of information simultaneously available to it the organism can identify objects and events with invariant relationships calculated over information in various streams. Haptic, kinesthetic, and vestibular information about body and head position can be used to calculate the angle of regard between perceiver and objects of perception. Texture and foreground and background relationship can give information about distance. If the organism can construct a cognitive map of its environment then it has even more powerful tools for keeping track of the relationship between it and perceptual objects.

Such devices will calculate identities for objects where the various appearances it presents to the organism can be seen as projective transformations of some canonical form. But this is not adequate for understanding the identities which exist between, e.g., the acorn and the oak, the caterpillar

and the butterfly, the ugly duckling and the graceful swan. These are identities which persist through morphological changes. The acorn and the oak are of the same substance, but they have different forms, different morphologies. The language which maps different appearances into the same identity must be meta to the one which maps different appearances onto the same morphology.

More abstract examples of this mechanism would be the identity of coal and diamond which is established through the concept of carbon, the identity established between electricity, magnetism, radio waves, and light through the theory of the electromagnetic field, the relativistic interconvertibility of matter and energy. These are also examples of different morphologies being mapped onto the same identities. But the morphologies are of a higher order than those considered in the previous paragraph; they come from a higher level of cultural evolution. Each level of cultural evolution is a propositional reconstruction of the level before. The propositional construction which sets the upper limit of a level of thought becomes the object of physiognomic perception to the next higher level, thus setting the lower bound for that upper level. The physiognomies which form the lower bound to this upper level can then be mapped onto morphologies and the morphologies mapped onto identities. The upper level is now well on its way toward the propositional reconstruction of the lower level. As this propositional reconstruction begins to stabilize a still higher level can begin to form above it. And so the process of cultural evolution grows upon itself.

The conservation principles which regulate computation at each level constrain the mapping of morphologies onto identities. The detective and the intelligence analyst want to know how to map various events, morphologies, onto identities established by causal coherence. Karl Popper explicated the principle appropriate to the detective's problem. David Hays has offered a principle which might be adequate to the analyst's task.

We are now ready for Gibson's reality criterion. He is concerned with detecting the difference between real images and imaginings induced by dreams, drugs, or hallucinations. In Gibson's words (p. 257):

> ...The most decisive test for reality is whether you can discover new features and details [of an object] by the act of scrutiny. Can you obtain new stimulation and extract new information from it? Is the information inexhaustible? Is there more to be seen? The imaginary scrutiny of an imaginary entity cannot pass this test.

This criterion needs to be strengthened in a simple way. The object in question must have an identity. For if it is organized only at the morphological level, then further scrutiny of the object may yield the replacement of one morphology with another and hence of one object with another. Only with an object constituted as an identity can the appearance of a new morphology be interpreted as providing more information about one identical object. The magician plays on this identity criterion by presenting a sequence of morphologies which is so discontinuous that we are forced to contemplate an object whose identity seems to shift from moment to moment. That sequence of shifts is the envelope of a higher level object which the magician creates, the abstract concept of illusion.

This Gibsonian reality criterion can be applied to the objects which emerge at each level of cultural evolution. As each level is the propositional reconstruction of the one below, so each needs a reality criterion which is ultimately a reconstruction of the one Gibson described. That is real which is constructed according to the appropriate conservation principle and which yields new information under scrutiny. A scientific theory which hasn't been falsified will eventually be abandoned when it doesn't seem to lead to anything new, when the phenomena it uncovers and explains seem more and more to be just trivial variations on well known themes. And only those lines of double contingency exploration which yield new information under conservation of substance-function correspondence should be maintained.

One can imagine a metagramming system which can accommodate Gibson's reality criterion. I suspect, however, that this sort of judgment is less likely to submit to routinization than conservation checking. In any case it will certainly show the same growth sequence, only a little at first with more later on.

Having discussed conservation, propositional reconstruction of physiognomy, and Gibson's reality criterion, we are now ready for a more abstract characterization of metagramming, a characterization couched in terms of a search through an abstract space.

METAGRAMMING AND SEARCH SPACES

Let us treat any cognitive activity as a search through some abstract space for a solution to the problem in question. We may regard each point in the space as containing the solution to some cognitive problem. We want to find the point which contains the solution to our problem.

Much work in the emerging field of cognitive science proceeds on just such a basis. The search space is generated using fundamentally propositional techniques. One can imagine that the points in such a search space are equidistant from one another. That is, problems and their solutions are uniformly distributed in propositionally generated search spaces.

However, when we superimpose a map of interesting and useable problem solutions on the space we find that these solutions tend to cluster in irregular regions in the space. The solutions which interest us are not distributed through the search space in any way which is neatly expressible within the propositional formalism used to generate the space. There simply is no reasonable way of using the properties of the space to identify those regions which are relevant to the problem at hand. The formalism which generated the space cannot be used to restrict search through it in a meaningful way. A metalanguage is needed to express the restriction.

Consider, once again, the problem of identifying photographs. The propositionally generated search space is one containing all possible visual objects. Imagine that in one region of the space we have a line which has a perfect circle at one end and a perfect square at the other. Between these points we have a continuum of figures somewhere between circle and square. We

have another line which goes between square and long thin rectangle (so long and thin that it is just a line); between these end points are rectangles whose length-width ratio varies between 1 and infinity. Just continue this procedure with all the simple geometrical figures you can imagine; and then continue it out until you get to relatively complex figures, like trees, and birds, and fish.

Now, if your pack of photographs resembles those which float around in my family, then most of the objects in the search space I've asked you to imagine simply do not exist in those photographs. There are a lot of human faces, but they all lie within a relatively limited region of the space. There will also be some dogs; and they too will lie within a limited region of the search space. But the region between human faces and dog faces will, for the most part, be completely unrepresented in the photographs. We simply do not have to be able to recognize any of the vast number of possible visual objects between dog faces and human faces. But conventional computing techniques for dealing with this sort of problem, which is one of pattern recognition, start with the full search space and have you compare the object to be recognized with each point (object) in the space. As a result much computing time and space is wasted.

To be fair, it should be said that no attempt is really made to search the entire space (except in the rather special case of brute force chess programs). Rather, the search is restricted to certain regions of the space. But these regions are identified in terms of the propositional structure of the space, which is not a very good way of identifying the useful regions of the space, since those regions will inevitably turn out to be very irregular under any set of propositional procedures used to generate the space. In a metagramming system the search space restriction is specified by a metalanguage which is linked to the object language through physiognomic representation functions, which are good for identifying complex irregular objects. In the metagramming M form (see pages 30-31) each Function-Information pair specifies a physiognomic index in a three-dimensional physiognomic search space. The application function establishes the identity of the object as a function over the three indexes and the current registration. The new value, the new identity, is then placed in Registration.

Now let's consider an example, chess. It is well known that a complete propositional strategy is available for playing the game. We know how to construct a space which contains every possible chess game and we can tell a player how to pick moves so that he'll never do worse than a draw. The only trouble is that explicitly constructing that space is impossible given any computer that we can imagine constructing.

This search space is generated by the rules of chess, which specify the moves each piece can make and the conditions which end the game. And the best strategy can be specified without rising above the space in some metalanguage. But the vast majority of possible chess games are never played--how many games open with any piece other than a King's or Queen's pawn? Yet all those games are possible chess games and all have their positions in the search space for all chess games.

We know in fact that chess players do not play chess in terms of deductions from the rules of chess--which is how you generate the search space. They play chess in terms of configurations of pieces on the board--excellent players seem able to recognize on the order of 50,000 such configurations. And those board positions cannot be readily generated as deductions from the rules of chess. One cannot state a compact deductive procedure for generating all and only those board positions in terms of which experts play chess. Those positions must be described in a metalanguage. Chess is played in terms of rules of board position and those rules are not deductions from the rules of legal piece movement and conventions for ending the game; those rules are about configurations of pieces describable within the rules of chess.

One plays chess in terms of weak and strong positions, exposed flanks, pincer movements, attack and retreat, defense and offense. But none of these terms can be defined deductively from the rules of chess--for example, try to state what it means for a piece to be under attack using only the rules of piece movement and capture. There is no way to remain within the rules of chess and define sequences of moves which will always strengthen one's board position. The writers of chess books have known this for years and so they convey chess strategy by extensive use of example. Any position which is like this one is, more or less, strong; while any position more or less like that one is weak. The number of examples needed to convey significant knowledge is quite large; but it is a finite number. And, it is the only way of dealing with the problem known to work well.

Programming a computer is much like chess. The rules of a given programming language can be used to generate all the possible programs writeable in that language. But most of those possible programs have never been written and need never be written; they are of no value. And programs are not written as deductions from the rules of the programming language. The machine won't accept programs containing syntax errors any more than one's opponent in chess will accept illegal moves. But that is only a minimal constraint on the code the programmer can write. And the rules of the language give no guidance in the problem of figuring out which sequences of code will perform the job at hand. The rules which tell programmers how to employ programming languages in solving computational problems are meta to the rules which define those languages. Some of these rules have been systematized in the principles of structured programming and applicative programming. But this doesn't yet get us to metagramming.

For the rules of programming in some language assume that the problem has been stated in computational form. Producing that statement requires a translation from a natural language statement of the problem to a statement in terms of data structures and algorithms. The rules for this translation haven't yet been systematized. That systematization is what metagramming is, rules for the transformation of problems stated in a natural language into a computational form. The rules involve a relatively small number of relations which are defined over relatively large sets of objects. Those objects are, on the one hand, informatic tasks to be performed, and, on the other hand, computational forms for realizing those tasks. The informatic tasks are interlinked with the computational forms at all levels with the forms providing

a means for regulating the tasks in the way syntax provides the means for manipulating semantics.

I want to conclude this section with the following analogy: The conceptual structures by which computers are currently designed have the same relationship to the conceptual structures needed to program them effectively as the rules of chess have to the rules of board position and strategy by which the game is played. As a mechanical device the computer has a functional form generated in the conceptual universe of Turing and von Neumann. The form is realized in an electromagnetic substance according to the principles of microelectronic engineering. But when you turn the machine on it displays behavior which cannot be adequately characterized in a conceptual universe whose upper limit is set by the principles used to design the computer. A reliable understanding of how to write software must be formulated in a language which is meta to the one in which the computer was designed. Computer science has been straining to rise to the task; but so far it has failed, leaving only a hopeless babble of mutually incoherent programming languages. The reason the metagrammer ranges from perceptual psychology, through epistemology and cultural evolution, is to construct a conceptual universe which is meta to the one in which the computer was originally designed. For it is only in this universe that computer programming can be done as well as humans currently play chess.

CONCLUSION: METAGRAMMING FOR INTELLIGENCE

We might say that metagramming works by inducing an ecological closure over an otherwise unbounded search space. The search space is generated by a propositional object language. The ecological closure is formed by a metalanguage which is linked to the object language through physiognomic specification.

In the M form, one of the Function-Information slots contains an account of the machine for which the metagramming system is generating code. That account can be used to generate all the possible programs which can be written for that machine. The problem is to select the program needed for the job at hand.

The ecological restriction on that domain is specified in the Function-Information slot devoted to the structure of the discipline, in this case, intelligence analysis. The third Function-Information slot handles the physiognomic-propositional conversion necessary to fit the object machine into its informatic environment; this slot thus embodies an abstract theory of computation which is indifferent to the substrate in which the computation is carried out, whether it is neural wetware (the structure of the discipline as it exists in the analyst's mind) or microelectronic hardware and software (the object machine).

The Application function coordinates the Functions in the three Function-Information slots so that the contents in Registration converge on a viable solution.

Once the initial system has been created much of the subsequent growth of the system would be regulated by the community of analysts who use it. One can imagine dialects and even idiolects growing up as individuals and groups adapt the system to their needs. Perhaps thorough reorganization will be periodically necessary. But as long as the conservation principles and Gibson's reality criterion regulate the reorganization the functional integrity of the system will be maintained. And through that the analyst's grasp on international reality will remain deft and firm and thus able reliably to guide the conduct of national policy.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


D
B2